CMS IN 2000/052

The CMS Interactive Graphical User Analysis (IGUANA†) "Functional Prototype" Software

Status Report

describing the IGUANA deliverables
for the

"Functional Prototype" Software Milestone
for the CMS
User Analysis Environment

31 October 2000

George Alverson, Ianna Gaponenko, Lucas Taylor††, and Lassi Tuura
Northeastern University, Boston, USA

† See also: `http://iguana.cern.ch/`
†† Coordinator/contact person

Executive Summary

This document summarises the status of the CMS "Interactive Graphical User Analysis" (IGUANA) software project. IGUANA was initiated in May 1999, as part of the CMS Software and Computing Project, to develop a coherent strategy and software base to address the medium and long term needs of CMS, as follows.

IGUANA Mission Statement

The IGUANA software project addresses interactive visualisation software needs for three domains:

- *graphical user interfaces (GUI's);*
- *interactive detector and event visualisation; and*
- *interactive data analysis and presentation;*

for use in a variety of areas such as offline simulation and reconstruction, data analysis, and test beams. Tasks include the assessment of use-cases and requirements and the evaluation, integration, adaptation, verification, deployment, and support in the CMS environment of visualisation software from HEP, academia, the public domain, and the commercial sector. Pre-existing software is exploited as much as possible to optimise the use of the resources available.

IGUANA constitutes one of the key components of an effective user analysis environment, the "Functional Prototype" for which is now completed as required for the corresponding CMS software milestone of October 2000. Software for this milestone must verify a baseline of technology choices though a set of working software prototypes which span a range of physicists' needs. It does not provide the complete functionality that is ultimately required nor does it necessarily satisfy stringent performance and quality criteria; this is a task of the next ("Fully Functional Software") development phase.

As reported to CMS and the LHCC, this "Functional Prototype" milestone was delayed by four months until the end of October 2000 for two reasons: firstly there was a shortage of one software engineer and secondly it was deemed desirable to align this milestone with the first prototype release of the Lizard interactive analysis software in October 2000 by the CERN/IT/API group.

The IGUANA deliverables for this milestone are:

1. documentation of the main **Requirements for the IGUANA "Functional Prototype"**;
2. a suite of documented **Software Prototypes and Packages** that satisfy the requirements;
3. a coherent **Software Infrastructure** including a software repository, multi-platform build, release, distribution, and documentation systems, and public versioned releases of both software and documentation.
4. a proposal for a **Baseline Choice of Visualisation Technologies** on which to start building the next phase of software;
5. a plan describing the **Proposed Project Evolution**.

The body of this document describes these deliverables and concludes that in terms of interactive visualisation, this milestone has been satisfied. The appendix provide more details of the software components.

For the next "Fully Functional" software phase it is proposed that the IGUANA development continues smoothly, consistent with CMS needs, for: "graphical user interfaces" and "interactive detector and event visualisation". IGUANA developments in the area of "interactive data analysis and presentation" are proposed to concentrate on CMS-specific data browsing aspects, which are complementary to generic activities elsewhere such as Lizard.

In this next phase the IGUANA project planning will be refined regarding: the identification of a more complete set of use-cases and requirements; the specification of the various tasks; estimations of the resources required; and specifications of the schedule, milestones, and associated deliverables taking account of the actual availability of resources and the priorities of CMS.

Contents

List of Figures

Chapter 1

Requirements for the IGUANA "Functional Prototype"

This section describes the first deliverable for the IGUANA contribution to the CMS "User Analysis Environment Functional Prototype" milestone of October 2000, which consists of a description of the main user and developer requirements.

These requirements will be refined and formalised further for the next phase of development following the milestone. This next phase will lead towards the "Fully Functional" (i.e. pre-production) analysis software milestone of December 2002, coincident with the CMS Software and Computing TDR.

1.1 Overall Aims and Scope

As described in the executive summary, the mission statement of IGUANA is:

IGUANA Mission Statement

The IGUANA software project addresses interactive visualisation software needs for three domains:

- *graphical user interfaces (GUI's);*
- *interactive detector and event visualisation; and*
- *interactive data analysis and presentation;*

for use in a variety of areas such as offline simulation and reconstruction, data analysis, and test beams. Tasks include the assessment of use-cases and requirements and the evaluation, integration, adaptation, verification, deployment, and support in the CMS environment of visualisation software from HEP, academia, the public domain, and the commercial sector. Pre-existing software is exploited as much as possible to optimise the use of the resources available.

It is worthwhile to expand on several aspects of this statement.

Firstly, IGUANA aims to support a generic and coherent set of tools which enable a wide variety of applications to be developed by both experienced and non-expert CMS software developers. For example, a physicist in a remote institute should be able to interactively view some ORCA events stored in an HLT Objectivity federation using an "out-of-the-box" application from IGUANA. Or, a developer of a test beam control system should be able to reasonably quickly create a new graphical user interface using IGUANA components and use it to monitor and control such parameters as high voltage settings and to view histograms in real time.

Secondly, IGUANA in no way attempts to re-invent the wheel; in fact the contrary is true. Wherever possible, the IGUANA developers exploit existing software from HEP, the public-domain, or commercial sector and ensure that it meets the needs and is well-integrated into a coherent overall environment. This strategy implies that

suitable software components must be ("small" and) modular with well-defined responsibilities and application programmers interfaces, rather than ("large" and) monolithic applications. This approach maximally leverages resources from outside CMS thereby allowing the relatively scarce resources available within CMS to concentrate on issues of integration and deployment, and development of the intrinsically CMS-specific components, such as the event display program.

Thirdly, IGUANA does not cover more general interactive graphics applications such as: web browsers, video-conferencing applications, CAD programs, visual programming tools, and so on.

The IGUANA project is an integral part of the CMS offline software systems, shown schematically in Figure 1.1, and the project planning [1]. Technology and design choices are made in consultation with other CMS software groups such that they are mutually compatible.

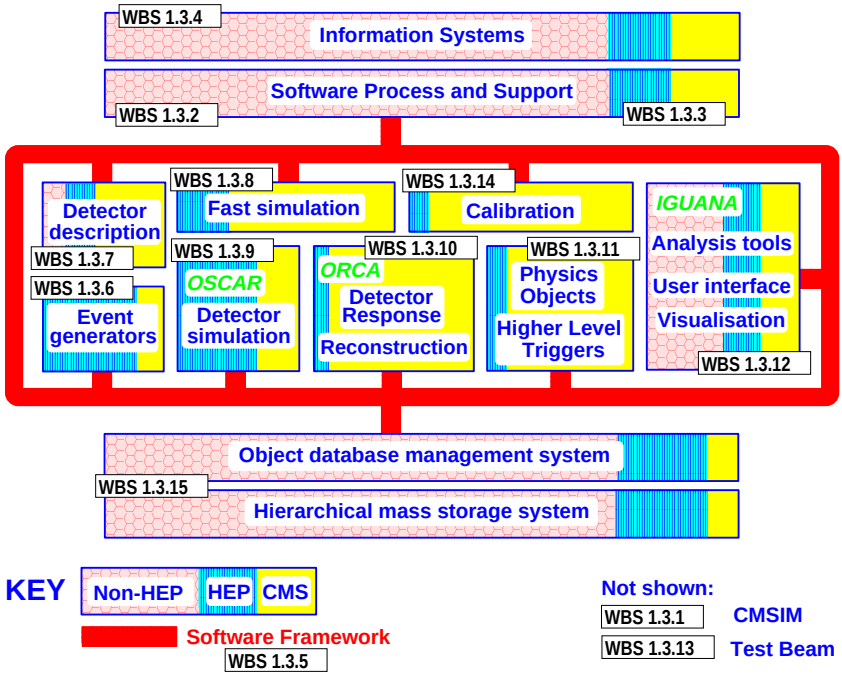

Figure 1.1: The modules forming the CMS offline software systems. The shading shows very approximately the relative contributions of non-HEP sources such as commercial vendors or open software consortia, HEP-wide efforts, and the CMS collaboration.

1.2 Modularity, Standards, and Leverage

Prior to a detailed discussion of the functionality required, we consider some general requirements on the software. It is crucial that the code can be maintained with modest amounts of effort on a variety of platforms and with varying external software dependencies, both centrally at CERN as well as in the institutes throughout the collaboration. For optimal use of resources, notably manpower, the software architecture should be consistent with significant re-use of the code at various levels of granularity within the same application as well as for similar tasks performed by distinct applications. A closely related requirement of the software is the need for continuous evolution throughout the lifetime of the experiment. This ability to evolve is required to insure against obsolescence, to satisfy the changing needs of the experiment without undue delays, and to remain competitive by being able to exploit new developments in software and computing.

These requirements in turn imply the following are central to the IGUANA (and indeed the CMS) software strategy:

- **Software Modularity**
  The code should be as modular as possible with well-defined interfaces between modules. This greatly facilitates the software maintenance, reuse, and ability to evolve. By ensuring that the code comprises loosely-coupled modules, one avoids the intrinsic non-scalability and non-maintainability of tightly-coupled monolithic systems.
- **Use of Standards**
  Widely-used standards, either *de-facto* or *de-jure*, should be adhered to as much as is reasonably possible to reduce dependencies on proprietary interfaces and to allow cost-effective public-domain and commercial software solutions to be exploited more easily. This will greatly facilitate integration, porting, and support in general and, by remaining in the mainstream, help to CMS to ensure that the software is able to evolve smoothly.
- **Leverage of Effort**
  Given the previous two central aspects of the IGUANA strategy immediately permits a great deal of effort from beyond CMS to be exploited. Modular software with well-defined interfaces can be easily imported and deployed from HEP projects *which follow the same design philosophy* as well as from public-domain projects or the commercial sector.

1.3 Graphical User Interfaces

The "Functional Prototype" requirements for the first IGUANA domain, "Graphical User Interface" (GUI), apply to the underlying software to be used to create a variety of GUI's for differing applications throughout the CMS software environment. Thus, the need to be easily re-used and easily integrated with other CMS code for a wide variety of situations, implies the following requirements:

- object-oriented extensible GUI toolkit;
- C++ Application Programmers Interface (API);
- a comprehensive set of standard widgets; and
- comprehensive documentation and working examples.

The need for the GUI software to be useful in the context of other graphical drawing software implies:

- compatibility of the GUI software with other 2D and 3D rendering software choices (required);
- support within the GUI software for high-level 2D drawing classes with off-screen rendering (preferable but not required); and
- compatibility with legacy Xt and Motif software (preferable but not required).

The need to support GUI applications on a variety of CMS platforms throughout the collaboration implies:

- cost-effective license (preferably free);
- support on a wide range of Unix flavours including Linux and Solaris (required); and
- support on Windows9x/NT/2000 (preferable but not required).

In addition to these reasonably well-defined requirements there are a number of characteristics which were taken into account in a less rigorous fashion. From the user point-of-view, the GUI should be self-consistent and resemble GUI's of other applications with which the user is already familiar by conforming to the prevailing look-and-feel standards. The GUI should enhance the usability of the application by minimising the learning time, reducing errors, and increasing the efficiency with which complicated command sequences can be executed. Generic commands, (*e.g.* copy, cut, paste, delete, or move) should perform the intuitively expected operation, *i.e.* the GUI should conform to the *Law of Least Surprise* in which the result of an action matches the expectation of the user. The GUI should provide support for varying levels of user expertise. A beginner should be able to find all desired actions graphically (e.g. using the mouse) while the expert should be permitted to use keyboard accelerator sequences for common operations such as copy, paste, or delete. While such features were not treated as formal requirements, they were taken into account when assessing various possibilities.

1.4 Detector and Event Visualisation

The "Functional Prototype" requirements for the second IGUANA domain, "Detector and Event Visualisation", apply to both the underlying software packages as well as the specific applications developed within CMS. The focus of this domain for this "Functional Prototype" milestone was an interactive application for visualising the CMS detector and the associated simulated and reconstructed event data.

As a general principle we aim for the well-known and successful Model-View-Controller (MVC) Architecture in which the visualisation is divided into three largely independent parts:

- the **model**, which constitutes the underlying data to be visualised;
- the **view**, which is the representation of the data on a display; and
- the **controller**, which handles interactions between the user and the model and/or view.

The requirements for these three aspects are described below.

1.4.1 Model

The model constitutes the underlying data to be visualised. Given the relative maturity of the GEANT3 simulation (CMSIM) compared to that of GEANT4 (OSCAR [2]), it was decided to base developments on the GEANT3 detector description at this time (GEANT4 will follow in future). In terms of requirements relating to the detector components to be visualised, this implies:

- the ability to read and display *arbitrary* elements of the GEANT3 CMS detector description, in particular the following generic volume types in the world coordinate system: BOX, TRD1, TRD2, TRAP, PAR, TUBE, CONE, TUBS, CONS, PCON, and PGON; and
- the ability to read and display *specific* elements of the CMS detector, notably for:
  - Tracker (Silicon wafers of: pixel barrel, pixel endcap, silicon barrel, silicon endcap);
  - Calorimetry (ECAL barrel rows, barrel crystals, endcap crystals, pre-shower, HCAL barrel towers, HCAL endcaps towers, tail-catcher, very forward calorimeter);
  - Muon Systems (barrel drift tubes, barrel RPC's, barrel absorber, endcap CSC's, endcap RPC's, endcap absorber);
  - Infrastructure (beam pipe, magnet).

The most important event visualisation application is that of simulated CMS data and the objects which are subsequently reconstructed by the ORCA reconstruction program. This implies:

- the ability to display, together with the detector view, transient and/or persistent data using CARF including the simulated data and the corresponding ORCA reconstructed data, in particular:
  - Tracker data (simulated hits, reconstructed hits, detector elements which contain hits, simulated tracks, reconstructed tracks);
  - Calorimetry data (ECAL reconstructed hits, ECAL clusters, HCAL reconstructed hits, ECAL + HCAL towers);
  - Muon data (simulated hits, reconstructed hits, detector elements which contain hits, simulated tracks, local track segments, reconstructed tracks).

Exactly how many of these will be available in the IGUANA "Functional Prototype" will depend also on the rate of progress of ORCA. In addition, many more items will ultimately be made available according to demand, for example vertices, electron candidates, jets, missing energy vectors, B-fields, neutron densities, dead and noisy channels, etc.

1.4.2 View

The view is the representation of the data on a display. The first question is whether one wants only 2D, only 3D, or both 2D and 3D. CMS is less symmetric than many previous detectors, for example there is only very approximate cylindrical symmetry in the CMS barrel region and only approximate azimuthal symmetry in general. This applies not only to the detector elements but also to the magnetic field, especially in the muon system barrel/endcap transition region. These considerations, coupled with the observation that acceptable, if not necessarily optimal, 2D views can be generated from slicing or projecting suitable 3D views implies the following view requirements:

- re-usable widget component into which arbitrary 3D views of the various models may be drawn;
- basic and performant image rendering including:
  - (very performant) wire-frame drawing where sets of lines represent edges of faces for planar surfaces and edges of meshed triangles for non-planar surfaces;
  - (less performant) solid surface rendering allowing for hidden-line and hidden surface removal;
  - specification of colour of each element.
- more sophisticated (and perhaps less performant) image rendering including many of the usual range of features provided by modern graphics packages, (preferably) including features such as:
  - choice of orthographic and perspective views;
  - full ray-tracing allowing for effects of ambient and directional point and planar coloured light sources;
  - specification of optical properties of all elements such as reflectance and transparency, with a choice of algorithms (e.g. screen door, additive, blended, sorted object blended, etc.) to allow for varying hardware performances;
  - effects of ambient and directional point and planar light sources;
  - specification of several colour properties for each element, such as emissive light colour and ambient and specular response to incident light.

1.4.3 Controller

The controller handles interactions between the user and the model and/or view. The main controls of programs should include as a minimum:

- **Next event** – to step to the next event (although it is desirable to defer resource intensive reading or processing until actually required by the user for the display);
- **Quit** – to terminate the program gracefully, performing "end-of-job" tasks and closing files as required;
- **Help** – various levels of on-line help such as "About this program", invocation of a WWW-based user guide and reference documentation; context-sensitive "tool-tips" or "bubble-help".
- **Save** – to create a persistent representation of the image as a metafile, that is, one that can be manipulated afterwards with more functionality than e.g. a simple bitmap;
- **Print** – as a bitmap (preferable but not required since essentially the same functionality is trivially available as an external "screen-grabbing" application).

The controller should support the following geometrically-related operations in a performant fashion:

- interactive control of the view parameters, including:
  - rotations about predefined orthogonal axes and arbitrarily-oriented axes;
  - magnification (zooming in and out);
  - translations in the plane of the display;

    - view clipping planes arbitrarily oriented in 3D space.
- ability to select ("pick") an arbitrary item in the 3D display and perform actions meaningful for the item which was selected;
- ability to select an arbitrary item in the 3D display and to set the scene centre-of-rotation to be this point;
- the control of the view should be straightforward and intuitive and include simplifying value-added features (desirable but not absolutely required), such as:
    - view manipulation using intuitive control widgets in the GUI outside of render window;
    - view manipulation with the mouse in the graphics render window;
    - saving of current view parameters as new "home" set of parameters and subsequent restoration of these "home" view parameters;
    - continuous view animation ("spin").

The high degree of complexity of the CMS detector and events imposes requirements on the visibility control and performance of the views. Required features are:

- the ability to perform data access or processing on-demand, such that no resources are expended for scene elements which are never actually visualised (for CMS event data this should be consistent with the CARF action-on-demand mechanisms);
- the ability to control the (in-)visibility of parts of the detector and event structures with arbitrary and varying levels of granularity;
- the ability to group graphics elements and to add or remove them from the graphics scene at run-time (e.g. to perform actions such as: "hide all the detector", or "show tracks and reconstructed hits superimposed on the simulated hits".

1.5 Interactive Data Analysis and Presentation

The "Functional Prototype" requirements for the third IGUANA domain, "Interactive Data Analysis and Presentation", cover four main categories:

- statistical and numerical analysis;
- histograms and ntuples/tags;
- plotting;
- interactive data browsing and analysis.

Prior to describing these in more detail (below) we emphasise the following points.

Firstly, the "Functional Prototype" does not include the full set of requirements for a complete system; rather these are a representative set of typical needs.

Secondly, to dispel potential mis-conceptions, we re-emphasise the IGUANA philosophy of integrating software modules from a variety of compatible in-house, HEP, public-domain, and commercial sources, rather than attempting to develop a complete in-house solution to the large domain of interactive data analysis and presentation.

Thirdly, the extent and boundaries of this domain are intrinsically ill-defined, both technically and organisationally. Therefore, the focus of this domain for the "Functional Prototype" milestone was on a broad suite of prototypes each of which demonstrated just one or a few specific features. The explicit plan following this milestone is to refine and extend the use-case analysis and requirements and determine the technical and organisational domain boundaries prior to significant further development.

1.5.1 Statistical and Numerical Analysis

The requirements for statistical and numerical analysis software may be loosely described as the need for roughly equivalent functionality as the CERNLIB's MINUIT and mathlib packages. These a very stable packages which have been used successfully throughout HEP for several decades. However, since they are Fortran/C-based and the original expertise involved in their creation is now almost lost, their long-term future support is not possible.

The requirements for the "Functional Prototype" milestone for statistical analysis software correspond approximately to requiring a modern and supportable replacement for MINUIT [3], with the following main features:

- the provision of a general-purpose minimisation engine (minimiser) for minimising arbitrary user-defined functions of a variable number (up to many tens) of parameters;
- the ability to minimise a comparable or preferably broader class of problems as MINUIT;
- the ability to perform minimisations with comparable or preferably better performance than MINUIT;
- the provision of OO/C++ API's suitable for both developers and end users;
- the ability to use the minimiser interactively or from a batch program;
- the ability to define the function to be minimised in the form of C++ code;
- the ability to choose between various minimisation strategies and algorithms;
- the ability to define the starting values of fit parameters;
- the ability to leave parameters free, to constrain them within bounds, or to set them to a fixed value;
- the ability to access the function value of the minimum and the associated fit and error analysis parameters;
- the ability to scan the shape of the function around the minimum as a function of one or two parameters;
- the ability to perform a standard error analysis of the minimum assuming a multi-variate Gaussian error distribution (inverted Hessian);
- the ability to perform a non-standard ("MINOS") error analysis of the minimum allowing for an arbitrary error distribution, based on the actual shape of the likelihood in the region of the minimum.

In addition to the requirements on the minimisation engine itself, it is desirable to have a value-added functionality for straightforward fitting of histograms, including:

- the ability to fit a histogram using pre-defined errors in each bin, or assuming the errors are Poisson-distributed;
- the ability to specify the fit function in terms of linear combinations of a few predefined functions, such as polynomial, Gaussian, exponential, etc.
- the ability to define the fit function in the form of C++ code;
- the ability to choose from several pre-defined minimisation algorithms/strategies;
- the ability to retrieve the minimisation results.

Additional requirements, also covering numerical analysis in some detail, have been previously produced as part of the LHC++ assessments [3, 4]. These may be roughly summarised as already stated above:

- the provision of modern and supportable software with approximately equivalent functionality to the Fortran/C-based mathlib library.

1.5.2 Histograms and Tags

The requirements on histograms for the "Functional Prototype" refer to their intrinsic nature and simple management operations, as follows:

- the ability to create a 1D or 2D histogram with arbitrary number(s) of bins in one (or two) dimension(s), upper and lower limit(s) on the independent variable(s), and associated descriptive meta-data such as title;
- the ability to copy a histogram to form a new, identical histogram;
- the ability to destroy a histogram;
- the ability to increment the contents of an arbitrary bin by an arbitrary amount, where the bin is defined by either an index or the value of the independent variable(s);
- the ability to fill all bins of the histogram in a single operation with a compatibly-dimensioned set of values;

- the ability to handle physically natural exceptions such as attempts to fill a histogram using out-of-range independent variables, and to keep track of common exceptions such as the integrated contents of underflow and overflow bins;
- the ability to perform operations on histograms such as reset, add, subtract, multiply, divide by scalar or by a histogram;
- the ability to accumulate and subsequently access statistics associated to a histogram;
- the ability to make histograms persistent.

External operations which merely *use* histograms, such as plotting or fitting, are not intrinsic responsibilities of the histograms themselves and are therefore described below.

Event tags combine and enhance features of ntuple and event directories. They contain summaries of key data used in analysis as well as associations to the more complete and complex data from which the tag was derived. A tag may even contain only the information relevant to the selection process itself, the data that is then used in the analysis being obtained from other objects "connected" to the tag via associations. Alternatively, it may contain both the data used for selection and that used for subsequent ntuple-style analysis. The exact role of tags in the overall data model will vary according to many factors. As a starting point, the "Functional Prototype" milestone was based on the following requirements, for the "Functional Prototype", of tags:

- the ability to create tags containing named ranges of variables of varying types, such as integers, floats, doubles, character strings, simple structures, and links to other data structures;
- the ability to set the values of the tag variables;
- the ability to make tags persistent;
- the ability to read a persistent tag and extract ranges of values of the tag variables and to resolve links to other data structures which are referred to from the tag;

Closely associated and required operations, which are not intrinsic properties of the tags themselves, include:

- the ability to scan a tag and obtain a textual representation of the data entries;
- the ability to project one or more attributes of a tag into a histogram;
- the ability to apply cuts on one or more tag quantities to support filtering of read, scan, and project operations.

1.5.3 Plotting

The task of plotting is predominantly an issue of data presentation and should therefore be de-coupled from underlying data formats. For example, a plot can legitimately be made from a simple vector, a histogram including statistical information such as error bars, or from a "project" operation acting on an ntuple or tag. The various graphics style and options were de-emphasised compared to the actual content of the plot, as reflected in the following plotting requirements, for the "Functional Prototype":

- the ability to render a plot to an area of the screen managed as a re-usable general-purpose widget;
- the ability to save a rendered image of a plot to a file, preferably in "vector" mode such as PostScript rather than only bitmap;
- the ability to draw axes and control key features such as range, increments, numeric labels, grid-lines, titles, and linear or logarithmic scaling;
- the ability to draw one or more 1D plots, on a set of axes, as (at least) connected lines, steps, marker points with error bars, and combinations of the above;
- the ability to draw one or more 2D plots, on a set of axes, as (at least) sets of markers (scatter plot), boxes scaled according to bin contents, contours corresponding to bins of equi-content, towers in the third independent dimension (lego plot), and coloured bins with the colour mapped according to bin contents;
- the ability to manage the layout of a (virtual) page in terms of independent zones, each of which may contain various superimposed plots;
- the ability to annotate plots with page, zone, plot, and axes titles, as well as arbitrary user text in user-defined locations;

- the ability to annotate plots with statistical information such as number of entries, plot integral, mean, r.m.s., number of under- or over-flow entries;
- the ability to change visual styles of graphics primitives such as lines, markers, fill areas, and fonts including sizes, colours, dashing, hatching, etc.

1.5.4 Interactive Data Browsing and Analysis

A data browsing and analysis environment bridges the experiments core software frameworks for reconstruction, analysis and data access, and the presentation and interaction components discussed so far to provide a productive and comfortable environment. What makes this particularly challenging is the wide range of tasks with no clear boundaries: from simple interactive analysis cuts on tags to reconstruction-like analysis to batch mode analysis to command line interaction to GUI interaction. Even just the operations on the data range from simple file-level tasks to interacting with individual objects.

In other words, the data browsing and analysis must respond to two fronts. On the one hand it must interact with the rest of the CMS software; this requires suitable interfaces between the interactive environment and the core frameworks such as CARF and ORCA (both batch and interactive modes). On the other hand, suitable presentation and analysis tools (such as fitting and plotting) must be available.

For the "Functional Prototype" phase the emphasis on this task has been on developing the visualisation and interaction toolkit and an architecture that allows evolution. As this task is very wide in range, it is difficult to write down a good set of requirements without a full use-case analysis. Given this and the fact that much of the functionality will be covered by various parts of the toolkit, we have opted to describe here a few use-cases for critical parts of the task as a whole. We expect to expand the use-case collection and develop a more complete set of requirements in the near future.

The general requirements we state so far:

- analysis and visualisation should not implicitly or explicitly determine application object design – they should have a minimal impact on it;
- no one visualisation system or method is or ever will be "the right one" because of varying needs and changing technologies, and hence the architecture should not be tied to a particular one but instead allow many to coexist and interact.

The use-cases the functional prototype must demonstrate are:

- demonstrate the ability to access, browse and visualise event, detector, and other data in the persistent store using the CMS CARF framework;
- demonstrate the browsing of persistent histograms;
- demonstrate the ability to browse tags in the database, to perform simply selection cuts on them, to plot the selection as histograms with user-defined binning selections and re-binning, and to fit functions to those histograms;
- demonstrate the ability to perform selections on tags using both simple C++ expressions and more complex functions that are dynamically compiled and loaded.

Chapter 2

Software Prototypes, Packages, and Documentation

This section describes the second deliverable for the IGUANA contribution to the "User Analysis Environment Functional Prototype" milestone of October 2000, which consists of a suite of software prototypes, packages, and documentation that satisfy the requirements described in section 1. These cover general aspects such as modularity, standards, and leverage, as well as the three top-level IGUANA domains:

- graphical user interfaces (GUI's);
- interactive detector and event visualisation; and
- interactive data analysis and presentation;

as described below and in more detail in the appendix. The software and associated documentation is available online at `http://iguana.cern.ch`.

An important aspect of the prototyping was the demonstration that the various technologies not only satisfied individual requirements but also that they could inter-operate amongst themselves effectively as well as with the wider CMS software environment (including CARF [5], ORCA [6], and SCRAM [7]), the software packages of Anaphe (formerly LHC++) [8], and various public-domain and commercial components.

2.1 Modularity, Standards, and Leverage

The general requirements, discussed in section 1.2, of *modularity*, use of *standards*, and *leveraging of effort* are reflected in the choice of the packages which form the user analysis software associated to IGUANA. Figure 2.1 shows these software packages, the functionality of which is described below. The arrows denote the inter-package dependencies determined using the *IgNominy* automated dependency analyser developed as part of IGUANA, as described in section 3.2.

Firstly, and supporting the modularity requirement, the figure shows that there are a number of loosely-coupled packages. Each package has a well-defined function and is on average directly dependent on only approximately two other packages. Thus, if a particular package fails and needs to be replaced or is simply superseded by a better package the vast majority of the software is completely unaffected.

Secondly, and supporting the use of standards, the figure shows the use of a number of *de-jure* or *de-facto* standards and widely-used packages adopted in HEP, academia and industry. These include X11 and OpenGL for the underlying low-level 2D and 3D graphics respectively; Qt and OpenInventor for the high-level 2D and 3D graphics respectively; as well as a commitment to the emerging set of AIDA (Abstract Interfaces for Data Analysis) standards for HEP analysis.

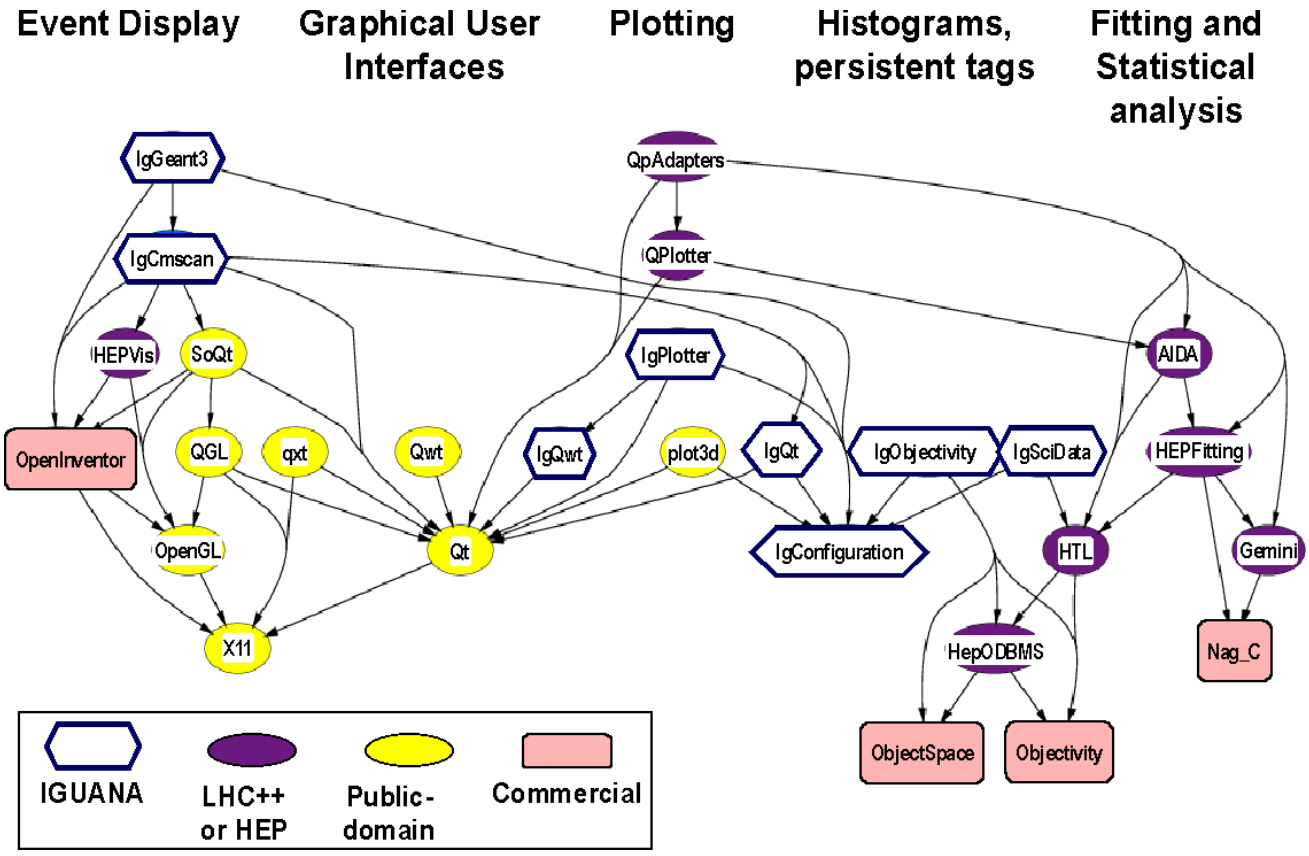

Figure 2.1: Software packages associated to IGUANA showing their inter-dependencies and origins. Application and example programs are not shown. An arrow from package A to package B denotes the dependence of package A on package B, *i.e.* package A cannot be used in isolation of package B. Note that applications and examples based on the toolkit are not shown in the figure.

Thirdly, and supporting the requirement to leverage non-IGUANA efforts wherever possible, the figure illustrates clearly the IGUANA strategy of exploiting high-quality software products developed elsewhere; roughly equal numbers of packages come from the four sources: IGUANA, LHC++/HEP, public-domain, and commercial sources.

2.2 Graphical User Interfaces

Figure 2.2 summarises the tasks, packages, their origins, and the historical and proposed future usage of software associated to the "Graphical User Interfaces" task. This task is further divided into sub-tasks, in particular "GUI Toolkits", "Widget Extension Libraries", and "Application Widgets" as described below.

2.2.1 GUI Toolkits

Although the underlying 2D graphics on Unix is universally X11, there was no single dominant GUI toolkit used by HEP. CMS has used the Motif library, for example for the "Cmscan Classic" event display program used with CMSIM. Motif is however hard to use and extend, is not written in C++ nor is particularly Object-Oriented, nor does it have high-level 2D capabilities. Although C++ wrappers do exist for various non-C++ widget sets

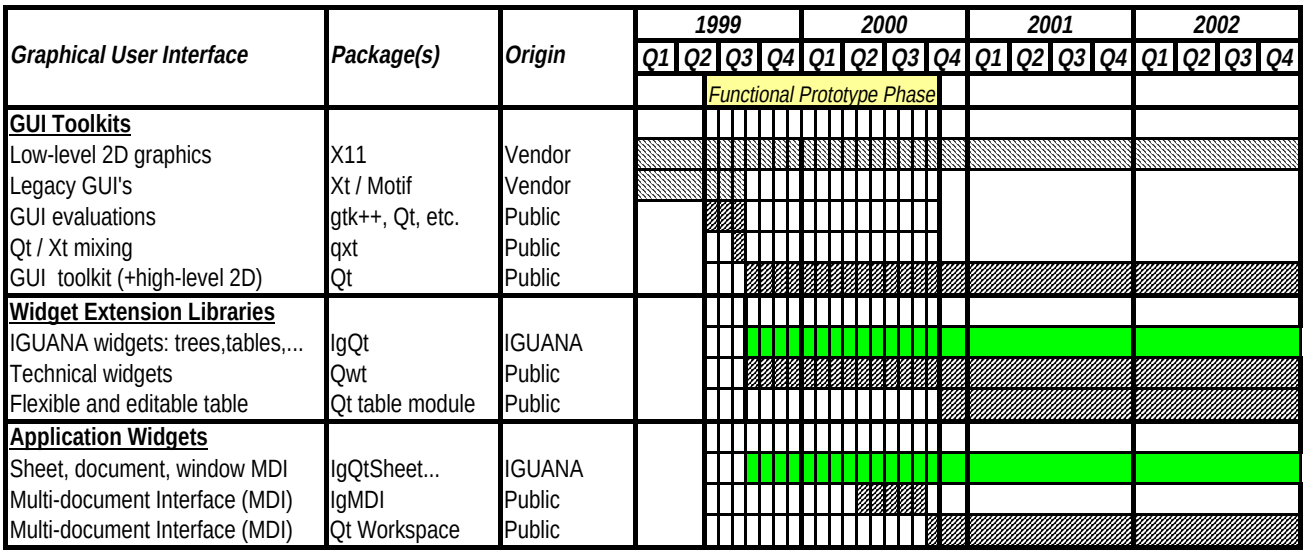

Figure 2.2: Tasks, packages, their origins, and the historical and proposed future usage of software associated to the "Graphical User Interfaces" task.

such as Motif, the functionality becomes the least common denominator of all the underlying window systems; and the performance is degraded by the extra layer.

Therefore, a range of GUI products were evaluated in the context of the requirements described in section 1.3. Given the plethora of existing GUI products, no serious consideration was ever given to the development of an in-house GUI solution as part of the IGUANA toolkit. Short-listed candidate GUI packages included: OpenAmulet; Qt; Coral; Fltk; FOX; GLUI; gtk+/gtk–; V; and WxWindows. Ultimately the Qt [9] toolkit from Troll-Tech was selected (Qt was subsequently adopted by Anaphe, the renamed LHC++ project). Qt is a C++ OO GUI toolkit providing a wide range of widgets, as illustrated in the IGUANA prototype interface shown in Figure 2.3(a).

Qt is available on a wide range of platforms and operating systems:

- Linux (Qt forms the basis of KDE, the "K Desktop Environment");
- Solaris, SunOS, HP-UX, Digital UNIX (OSF/1), Irix, AIX;
- Microsoft Windows 95/98 and Windows NT;
- FreeBSD, BSD/OS, SCO and others.

It is freely available for non-commercial development for all Unix platforms; for Windows, the developers license is approximately $1.5k. There are no run-time license costs nor royalties to be paid on any platform, irrespective of whether archive or dynamically-loaded libraries are used.

Qt widgets are treated as software components which communicate by emitting *signals* which may be received by *slots*. A signal, with arbitrary data, can be emitted if an object state changes and any number of slots (including none) can receive the signal data. There are no explicit call-back pointers to functions (as in e.g. Motif) which are exposed to the developer or user. This greatly facilitates the design and implementation of modular maintainable code. In addition to the C++ binding, Qt Perl and Python bindings also exist [10].

For full portability Qt also provides, and uses internally, template-based collections, file and a general I/O device, directory management, date/time classes, and regular expression parsing. The philosophy is to provide these classes as optional functionality although of course Qt-based applications can also exploit other implementations, such as those of STL.

Incidentally, it is interesting to note that it is also possible to convert Qt applications, using the LiveConnect extension for plug-ins [11], to run within a Web browser such as Netscape Navigator or MS Explorer.

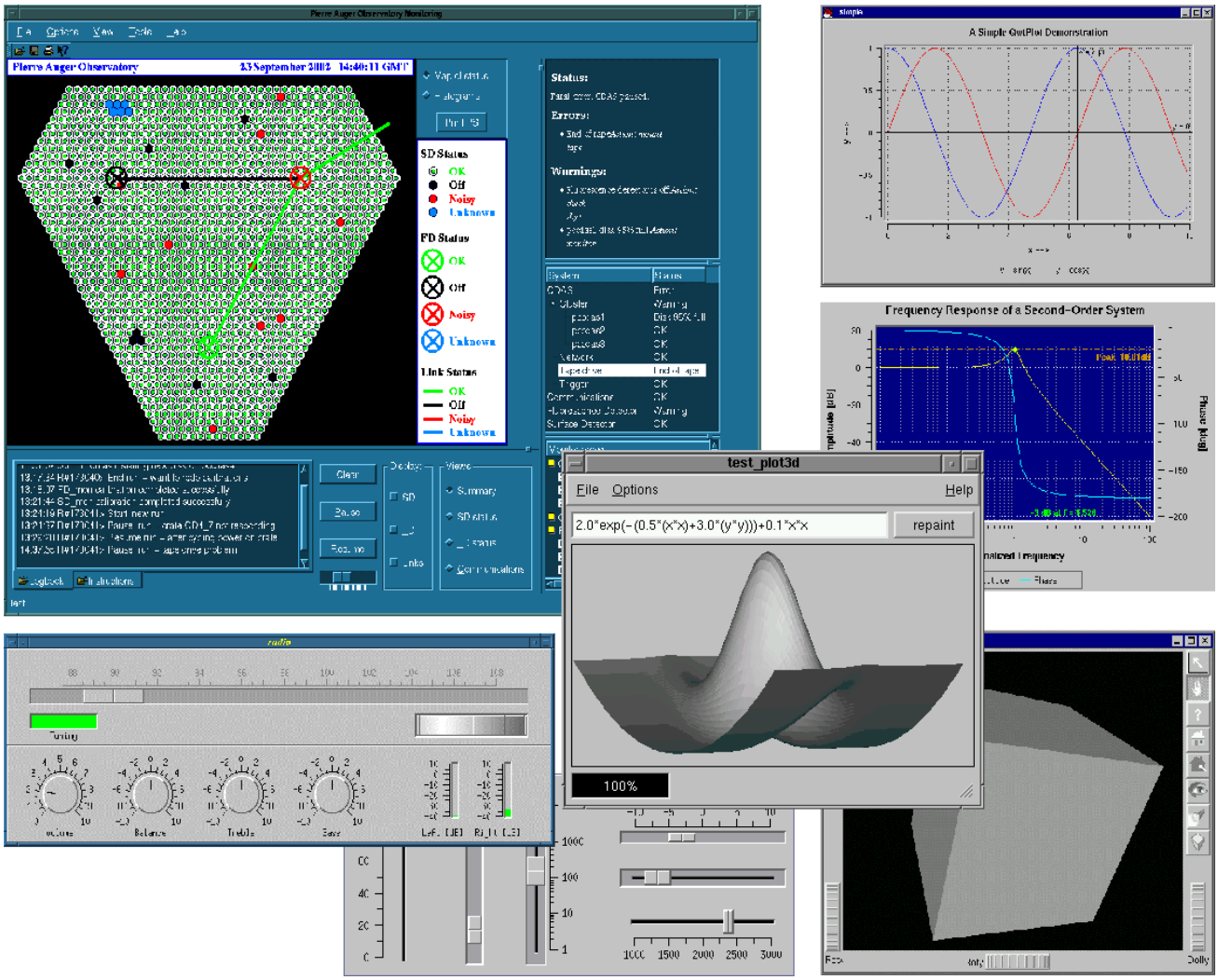

Figure 2.3: IGUANA Graphical User Interface prototypes demonstrating the functionality of the Qt library and various public-domain Qt-based extensions: a) basic Qt library; b) Qwt controllers; c) Qwt plotter; d) plot3d surface plotter; and e) SoQt 3D scene viewer.

2.2.2 Widget Extension Libraries

When choosing the GUI toolkit consideration was given to its extensibility and the availability of potentially useful public-domain extensions with particular relevance to IGUANA-related needs. The design of Qt was found to be well-suited for extending. Each widget is a C++ class which makes extending or over-riding the behaviour of widgets straightforward through sub-classing.

A number of public-domain extensions to Qt were evaluated and found to be useful for IGUANA, some examples of which are are shown in figures 2.3(b)–(e). These included instrument control widgets, simple histogram plotting widgets, 2D function surface renderers, and 3D viewers as described elsewhere. To facilitate distribution of such external software, it is imported into the IGUANA repository as part of the `Ig_Imports` sub-system. In addition to imported widgets, a number of extensions have been developed by the IGUANA team in the `IgQt` sub-system.

2.2.3 Application Widgets

"Application Widgets" refers to widgets which facilitate the building of applications in a coherent and homogeneous fashion. Initially a sheet-architecture was developed by the IGUANA team in which the main application window contained a number of layered "sheets" managed within a tabbed widget, similar to the way in which the MS Excel program manages multiple tables and charts from a single file. An example of the IGUANA implementation is shown in the interactive analysis example of Figure 2.9.

Approximately mid-way through the "Functional Prototype" phase, public-domain prototypes of Qt-based Multi-Document Interfaces (MDI's) started to become available. These support the creation of a main application window with a main menu-bar, tool-bars, and a workspace in which one or many "documents" are handled. Each document is represented within a separate widget which may be created, opened, closed, maximised, minimised (iconised), tiled with other documents, etc. A document may correspond to a separate file (e.g. as implemented for MS Word, MS PowerPoint, etc.) or may be one particular view of a file (e.g. MS Project).

Towards the end of the "Functional Prototype" phase, the very functional MDI implementation known as the "Qt Workspace Module"became available. This, together with some IGUANA-specific extensions, will form the basis for future IGUANA graphical application widget organisers.

2.3 Interactive Detector and Event Visualisation

Figure 2.4 summarises the tasks, packages, their origins, and the historical and proposed future usage of software associated to the "Detector and Event Visualisation" task. This task is further divided into sub-tasks, in particular "Underlying Graphics", "Viewers", and "Detector Interfaces / Shapes" as described below.

Detector&Event Visualisation	*Package(s)*	*Origin*	*1999*				*2000*				*2001*				*2002*			
			Q1	*Q2*	*Q3*	*Q4*	*Q1*	*Q2*	*Q3*	*Q4*	*Q1*	*Q2*	*Q3*	*Q4*	*Q1*	*Q2*	*Q3*	*Q4*
					Functional Prototype Phase													
Underlying Graphics																		
Low-level 3D graphics	OpenGL	Public/vendor																
High-level 3D graphics	OpenInventor	Vendor / free																
Qt / OpenGL mixing	QGL	Public																
Qt / OpenInventor mixing	SoQt	Public																
High-level 2D graphics	Qt Canvas	Public																
Viewers																		
Motif Detector / Event display	Cmscan	IGUANA																
Interactive 3D viewers	Ig3DViewer	IGUANA																
3D Detector / Event display	IgCmscan	IGUANA																
2D Detector / Event display		IGUANA																
Detector Interfaces / Shapes																		
GEANT3 interfaces	IgGEANT3	IGUANA													?			
HEP OpenInventor extensions	HEPVis	HEP																
GEANT4 interfaces	IgGEANT4	IGUANA																

Figure 2.4: Tasks, packages, their origins, and the historical and proposed future usage of software associated to the "Detector and Event Visualisation" task.

2.3.1 Underlying Graphics

The underlying graphics software is required, as described in section 1.4.2, to render (draw) a particular view of the underlying data model on the screen or to an off-screen file or printer. It does not cover the GUI controller software, which is described separately in section 2.3.2.

The low-level 3D graphics used for IGUANA is based on the OpenGL library [12], the dominant *de-facto* industry standard. The OpenGL rendering process is highly optimised in order to obtain satisfactory performance, in particular when dealing with many of the issues not described here [13], such as ray-tracing, hidden-line and surface removal, transparency, anti-aliasing, depth cuing, z-buffering, diffuse and specular reflection, refraction, shadows, texture and bump mapping, and colour mixing. In general, OpenGL is implemented in low-cost hardware accelerating graphics cards. Genuine 3-D images using stereo vision are possible, using two rendering operations for the cameras associated to each eye, although such possibilities have yet to be explored for CMS.

The high-level 3D graphics is based on OpenInventor [14], a high-level C++ class library based on OpenGL. OpenInventor provides optimised graphics *nodes* such as markers, lines, surfaces, volumes, text, materials, transformation nodes, lights, cameras, groups, etc. and mechanisms for creating, and manipulating the contents of hierarchical *scene graphs* containing many nodes. These may then be rendered, using OpenGL, to form the displayed image. OpenInventor provides considerable extra functionality in addition to these core tasks.

The OpenInventor implementation currently used for IGUANA is the commercial implementation from TGS which is licensed for all CMS collaborators through the Anaphe (formerly LHC++) project of CERN/IT. Several less robust/complete public-domain implementations of OpenInventor exist (e.g. Coin and SoFree) and it is of notable interest that the original source code from SGI for OpenInventor recently became public-domain. It is not unreasonable to suppose that one or several public-domain implementations may ultimately supersede the TGS implementation.

The ability to use the 3D graphics libraries, OpenGL and OpenInventor, together with the chosen GUI solution is a key requirement. The former was demonstrated within IGUANA using the public-domain "QGL" extension [15] which allows OpenGL windows to be created and managed as any other Qt widget. Similarly, the mixing of OpenInventor with Qt was achieved using the freely-available "SoQt" extension from "Systems In Motion" [16]. After some necessary modifications to these extensions this strategy was demonstrated to work successfully. This is shown in Figure 2.5 which illustrates the IGUANA-based viewer, deployed within ORCA, with an OpenGL render area in which 3D OpenInventor shapes are displayed and manipulated under the control of the surrounding Qt GUI.

2.3.2 Viewers

Having demonstrated that OpenGL and OpenInventor could inter-operate with Qt, the IGUANA team started to port their previous Motif-based 3D viewer to Qt. Two prototypes, Ig3DViewer and IgCmscan, were developed and ultimately merged into a single generic widget component known as IgCmscan. IgCmscan consists of an OpenGL graphics render area into which an arbitrary OpenInventor scene-graph can be rendered, as shown in Figure 2.5.

Associated to the render area are a number of controller features, the requirements for which are described in section 1.4.3. The pictorial tool-bar shows support for picking of arbitrary objects in the scene, the mouse view manipulations, setting and restoration of the home view, view-all, picking and centring on the selected object, and the choice of the camera (orthographic or perspective). Most of the features have the same layout as the familiar Xt/Motif-based Examiner viewer from TGS.

The requirements on the selection and adjustment of view-points are described in section 1.4.2. The view-point parameter controls (rotation, translation, and zoom) are available using just the mouse in the render area and also from intuitive controls surrounding the frame. The tree-widget to the right of the pictorial tool-bar

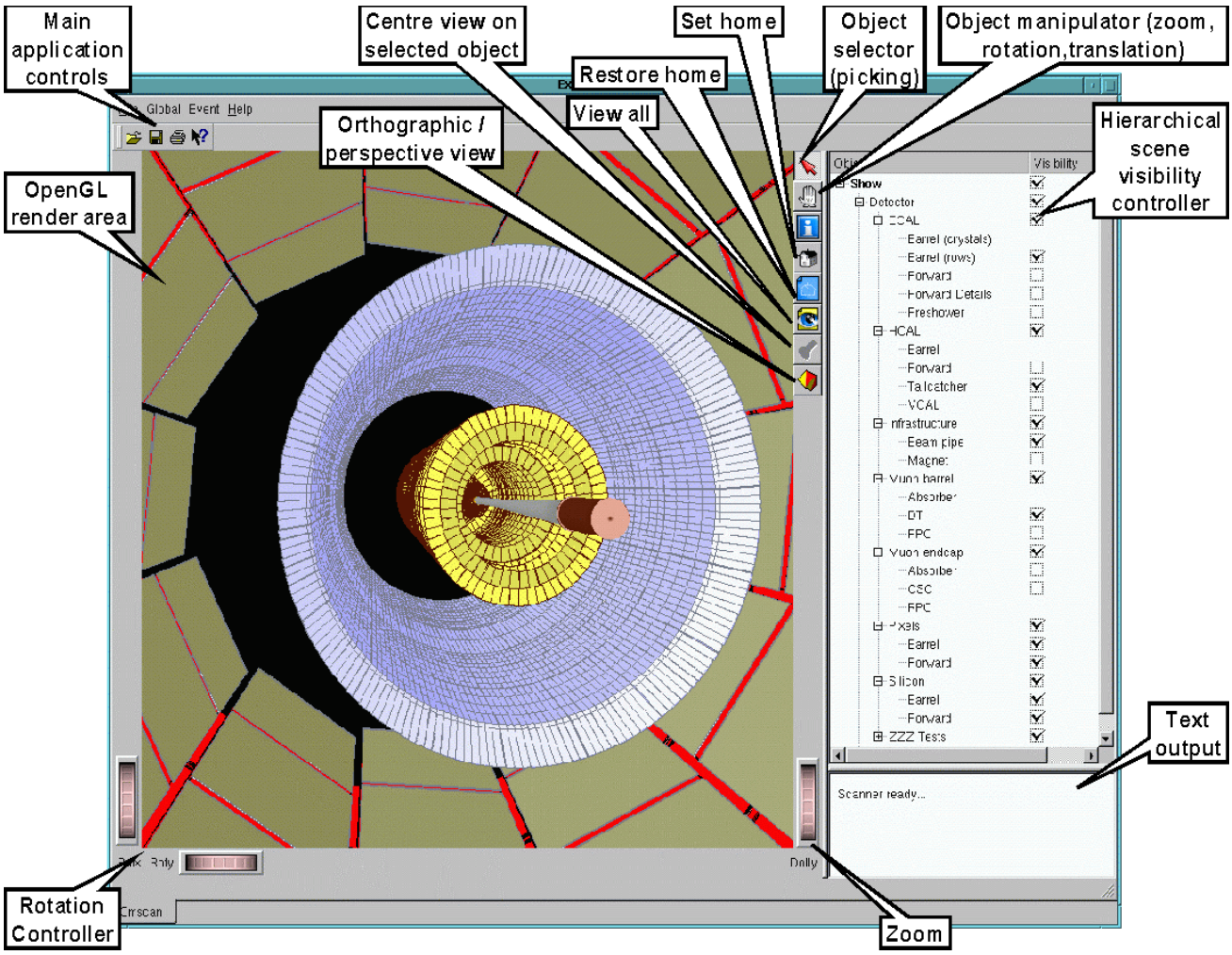

Figure 2.5: Example of the IgCmscan 3D viewer showing a view of the CMS detector.

is an IGUANA Qt-extension widget for controlling the visibility of arbitrary trees of drawable objects in an OpenInventor scene-graph.

Many less frequently used options and controls are available from a right-mouse pop-up menu, including: static and animated draw styles (solid, wire-frame, etc.); clip-plane depths; auto-rotation; seek animation parameters, and more.

2.3.3 Detector Interfaces and Shapes

This task concerns the ability to create drawable shapes from an independent detector description, the requirements for which are described in section 1.4.1. There are two aspects to this: firstly the interface to the underlying detector description and secondly the creation of graphics objects (OpenInventor nodes) which correspond to detector elements.

For the "Functional Prototype" the detector geometry used throughout CMS was stored in GEANT3 format. Therefore, IGUANA developed a generic interface to GEANT3, called IgGeant3. This provides an interface between the Fortran of GEANT3 and the C/C++ API of OpenInventor for the visualisation of completely arbitrary detectors described by a GEANT3 geometry description. It provides Fortran routines for walking down arbitrary GEANT3 geometry (GCVOLU common block) trees and requesting the creation of 3D OpenInventor

geometry transformations and scene objects (shapes). Either full or partial traversal of the tree is provided through the GSATT options which are also used to control with arbitrary granularity the visibility of standard, divided, or repeated volumes. C++ code is provided to create OpenInventor shapes in World coordinates for the GEANT3 shapes, as follows: `hv_gbox.cc` (for: BOX, TRD1, TRD2, TRAP, PARA); `hv_gcyl.cc` (for: TUBE, CONE); `hv_pcon.cc` (for: TUBS, CONS, PCON); and `hv_pgon.cc` (for: PGON).

In addition to the basic OpenInventor shapes such as box, cylinder, nurb-surface, etc., some more complex HEP-specific detector shapes were developed. These were fed into the HEP-wide HEPVis library which is subsequently imported into IGUANA for convenience of integration and distribution. It is worth noting that, through HEPVis, the basic GEANT4 volumes are available now as OpenInventor objects which should facilitate our future migration from GEANT3 with GEANT4.

2.3.4 ORCA Detector and Event Display Program

The sections above describe the generic components which have been evaluated, developed, and integrated within IGUANA to form the ingredients for generic 3D visualisation applications. The IGUANA developers have used this software to create the `Visualisation` sub-system of ORCA to provide a display program for the CMS detector and simulated and reconstructed event data according to the requirements of section 1.4.1.

Figure 2.5 shows the ORCA detector and event visualisation program interface and Figure 2.6 shows various other views of the detector and event data. The application is a CARF interactive application in which the event loop is controlled by the user through an IGUANA viewer. Detector data is accessed from GEANT3 using the generic IGUANA IgGeant3 interfaces to convert to OpenInventor objects. Simulated and reconstructed event data are requested from CARF which in turn causes it to be either computed on demand or accessed from the Objectivity persistent object store. An action-on-demand architecture is employed both at the level of the IGUANA scene-view controller as well as in the underlying CARF layer. Thus no processing is performed for any object until the user has positively requested to see it on the screen. Once a request is made the necessary data access requests are made and computations performed.

2.4 Interactive Data Analysis and Presentation

Figure 2.7 summarises the tasks, packages, their origins, and the historical and proposed future usage of software associated to the "Interactive Analysis and Presentation" task. This task is further divided into sub-tasks, in particular "Statistical and Numerical Analysis", "Histograms and Tags", "Plotting",and "Interactive Analysis" as described below.

2.4.1 Statistical and Numerical Analysis

The responsibility for satisfying the requirements for statistical and numerical analysis software, described in section 1.5.1, has been undertaken by CERN/IT. These may be loosely described as the need for maintainable OO/C++ software with functionality roughly equivalent to the MINUIT and mathlib packages (see also [3,4]).

As shown in Figure 2.7, mathlib has been replaced by the NAG_C mathematical library while the functionality of the MINUIT minimisation engine is replaced by that of the NAG_C minimiser. Given the importance of validating such crucial software, CERN/IT has provided a common set of interfaces: GEMINI in C and, on top of that, HepFitting in C++ as shown schematically in Figure 2.8(left). These interfaces enable MINUIT and NAG_C to be compared for a wide range of realistic physics problems. Figure 2.8(right) shows an IGUANA example IgFitting which demonstrates the use of HepFitting, GEMINI, and NAG_C to fit an HTL histogram and display it using an IGUANA histogram plotter IgPlotter.

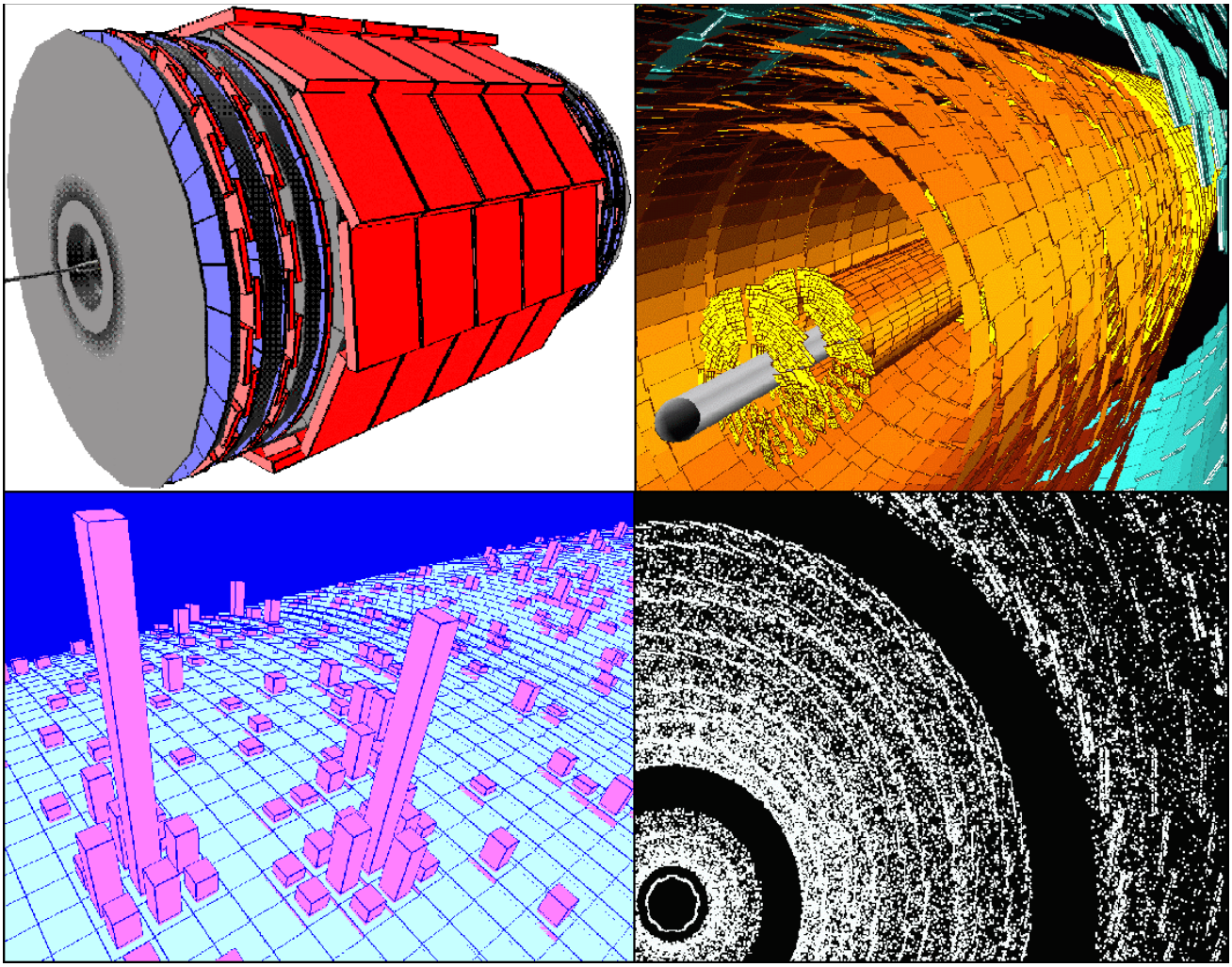

Figure 2.6: Views from the OrcaVis application showing: (a) the complete CMS detector; (b) the inner tracker; (c) energy deposits in the crystal calorimeter; and (d) raw hits in the tracker.

2.4.2 Histograms and Tags

The basic requirements on histograms and tags for the "Functional Prototype", described in section 1.5.2, are satisfied by the HTL and HepODBMS packages developed as part of the Anaphe (formerly LHC++) activities. It is expected that some steady evolution will be needed in future to satisfy the final requirements.

IGUANA prototypes verified that HTL allowed histograms to be created, copied, filled, destroyed, operated upon (scaled, added to other histograms, subtracted, etc.), and made persistent using Objectivity. For example, Figure 2.11 (upper) shows a prototype in which an IGUANA Objectivity hierarchical browser is used to locate HTL histograms in Objectivity and plot them using the plotting prototype Qplotter. External operations which merely *use* histograms, such as plotting or fitting, are not responsibilities of HTL. This functionality is covered by other packages which have also been prototyped and/or verified, as described in sections 2.4.1 and 2.4.3.

HepODBMS tags contain summaries of key data used in analysis, as might an ntuple. In addition, they support associations to the more complete and complex data from which the tag was derived. Various IGUANA prototypes demonstrated that HepODBMS tags could be created, filled, made persistent, and read back as expected. For example, Figure 2.11 (lower) and Figure 2.9 show IGUANA prototypes in which tag data stored in multiple

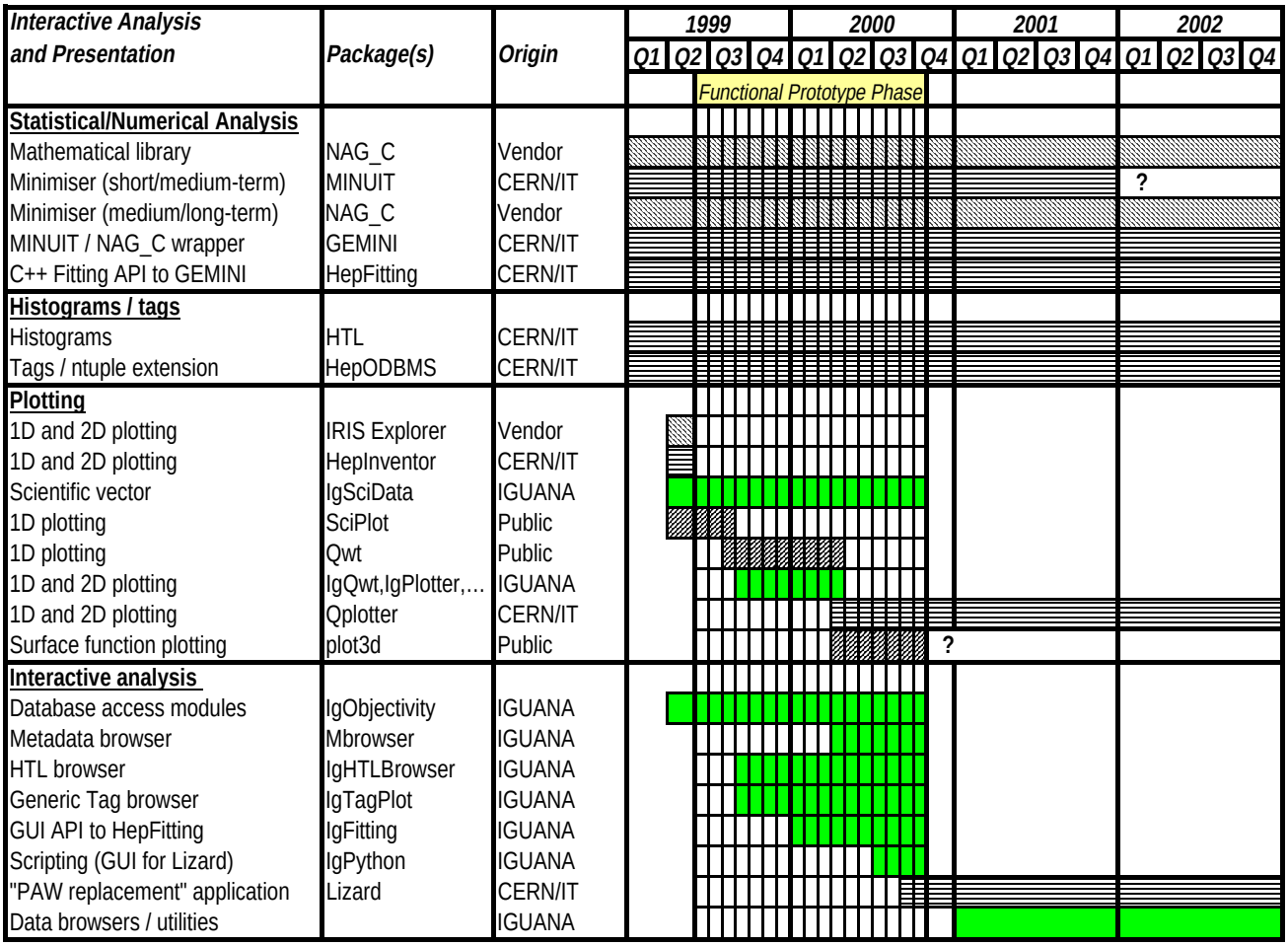

Figure 2.7: Tasks, packages, their origins, and the historical and proposed future usage of software associated to the "Interactive Analysis and Presentation" task.

databases of an Objectivity federation are browsed. The tag variables may then be displayed as 1-D or 2-D plots using an IGUANA plotter, with arbitrary (simple) cuts applied.

2.4.3 Plotting

Various possibilities for a reusable histogram plotter component were evaluated by IGUANA developers. These were based on a variety of existing software packages including HepExplorer / IRIS Explorer, SciPlot / Xt / Motif, and HepInventor / OpenInventor / OpenGL, as shown in Figure 2.10. These also demonstrate three different IGUANA prototype Objectivity browser widget components. These three were ultimately rejected for the following reasons respectively: framework rejected by users; non-OO Xt/Motif nature rejected by developers; use of a commercial 3-D package is overkill for relatively simple 2-D application.

The IGUANA team then decided to develop a simpler more coherent solution based on Qt and the public-domain Qwt library. This approach was found to have significant advantages. It is intrinsically OO, as is the HEPInventor implementation but in contrast to the SciPlot prototype which is not. It requires no external libraries other than Qt which is required already for the interface, in contrast to the HEPInventor implementation which requires the OpenGL, OpenInventor, and TGS MasterSuite libraries. In addition, the Qt solution has built in off-screen rendering to vector postscript whereas the others can only be rendered as bit maps. Figure 2.11 shows a Qt/Qwt-based IGUANA prototype in which a hierarchical browser is used to locate HTL histograms in Objectivity and

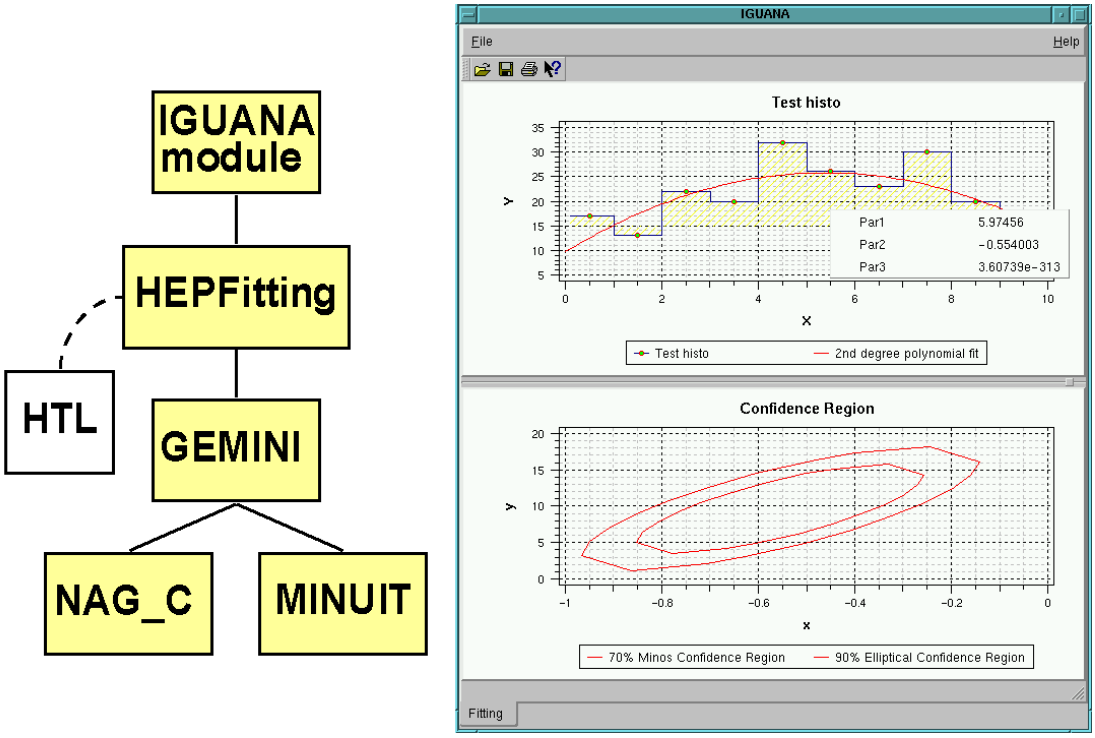

Figure 2.8: Fitting example showing the hierarchy of the minimisation and fitting software packages (left) and an IGUANA prototype displaying an HTL histogram, the result of the fit and the associated likelihood confidence contours (right).

plot them using an IGUANA plotter prototype.

Following this work, in mid-2000 the CERN/IT API group adopted the IGUANA plotter strategy. This has subsequently evolved, in the context of Anaphe, into the Qplotter package, with ongoing close collaboration with the IGUANA team but with CERN/IT commitment for support.

2.4.4 Interactive Data Browsing and Analysis

This task differs from many other IGUANA tasks in that it is mainly one of integrating many individual components together into a coherent whole. Since the emphasis of the IGUANA "Functional Prototype" was on building the toolkit, the requirements for this subtask, described in section 1.5.4, are less specific than for other tasks and merely comprise several typical use cases. The corresponding prototypes are listed in Figure 2.7 and described below.

IGUANA provides database access modules (IgObjectivity) that demonstrate the ability to access persistent HTL histograms and explorable HepODBMS tags stored in Objectivity. Tags and their attributes may be accessed by name and filtered according to simple user-defined cuts.

As shown in Figure 2.11(upper), the example (IgHTLBrowser) demonstrates the browsing of persistent histograms. The database structure is presented as a "file-system like" tree where a user can open and close a federated database, databases, or containers. He may then select a histogram stored in a database or a container and plot or print it.

Various generic Objectivity tag browser have been implemented as IGUANA sheet widgets that provides access to all the Objectivity tags within the database collections. Figure 2.11(lower) and Figure 2.9 show IGUANA prototypes in which tag data stored in multiple databases of an Objectivity federation are browsed. The tag

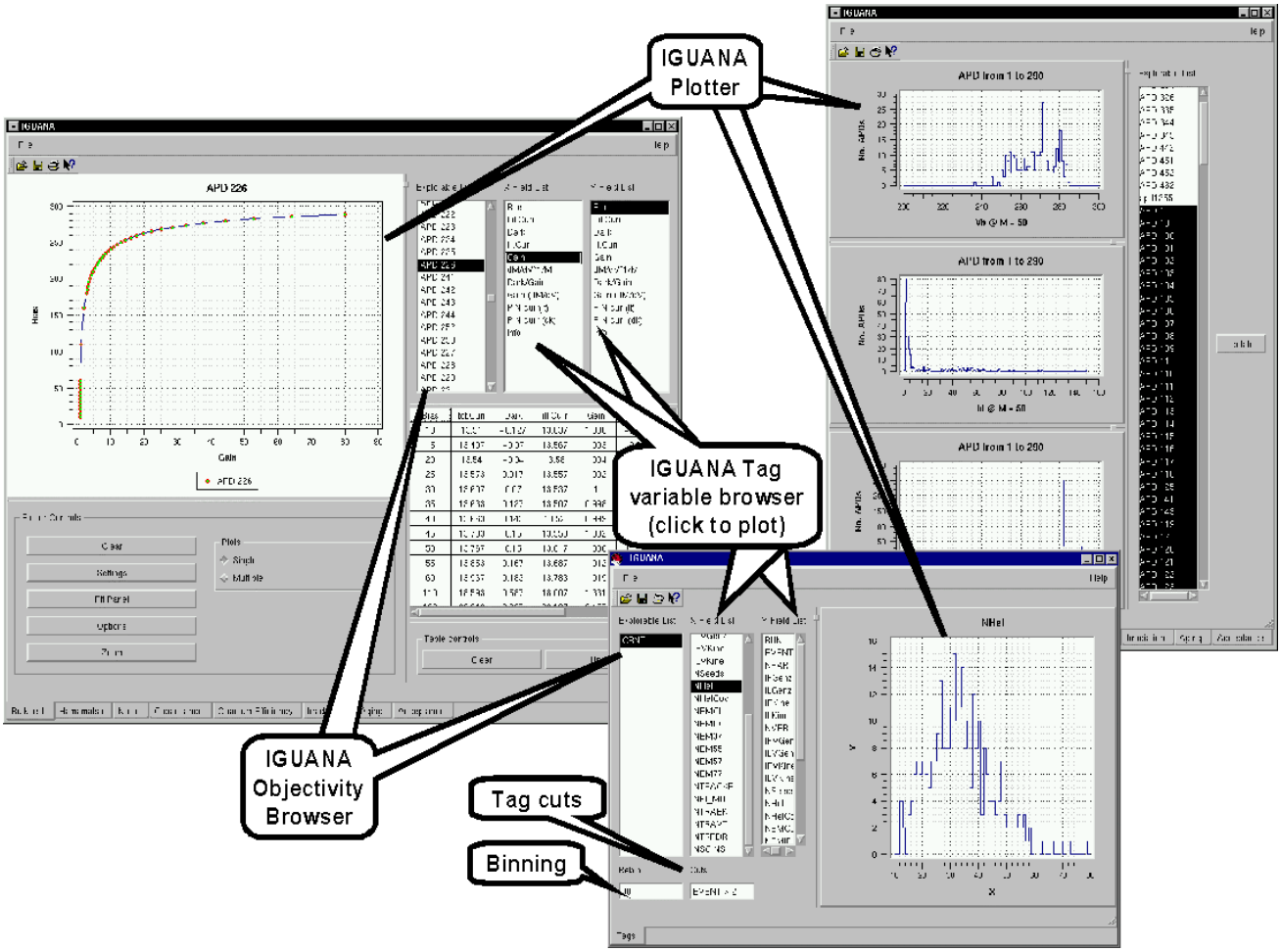

Figure 2.9: Interactive analysis with generic HepODBMS tags.

variables may then be displayed as 1-D or 2-D plots using an IGUANA plotter, with arbitrary (simple) cuts applied.

Going beyond tag browsing, an IGUANA meta-data browser (Mbrowser) has been prototyped which demonstrates interactive navigation of meta-data stored by CARF in Objectivity. For example, with ORCA data sets, this permits browsing of production event collection meta data information such as data set names, owners, and run numbers

As described in the previous section IGUANA has demonstrated the possibility to access, browse and visualise simulated and reconstructed ORCA event data using CARF as shown in Figures 2.5 and 2.6 .

Lizard is a CERN/IT project of Anaphe (formerly LHC++) for an interactive analysis environment based on AIDA specifications and a scripting interface using Python. Whilst currently providing limited functionality, Lizard should eventually cover all the basic features of PAW (and more). The first prototype release of Lizard made this October has the following main functionalities driven from the command line:

- create an HTL histogram and fill it;
- create a vector of points (N-dimensional points with errors) from a histogram and plot graphically on the screen (1D or 2D);
- define a fit model, add fit parameters, and fit a vector or a histogram;
- access data in an “ntuple” (based on tags), project an attribute into a histogram with or without a cut;
- on-the-fly compilation and dynamic loading of user-defined cuts and fits.

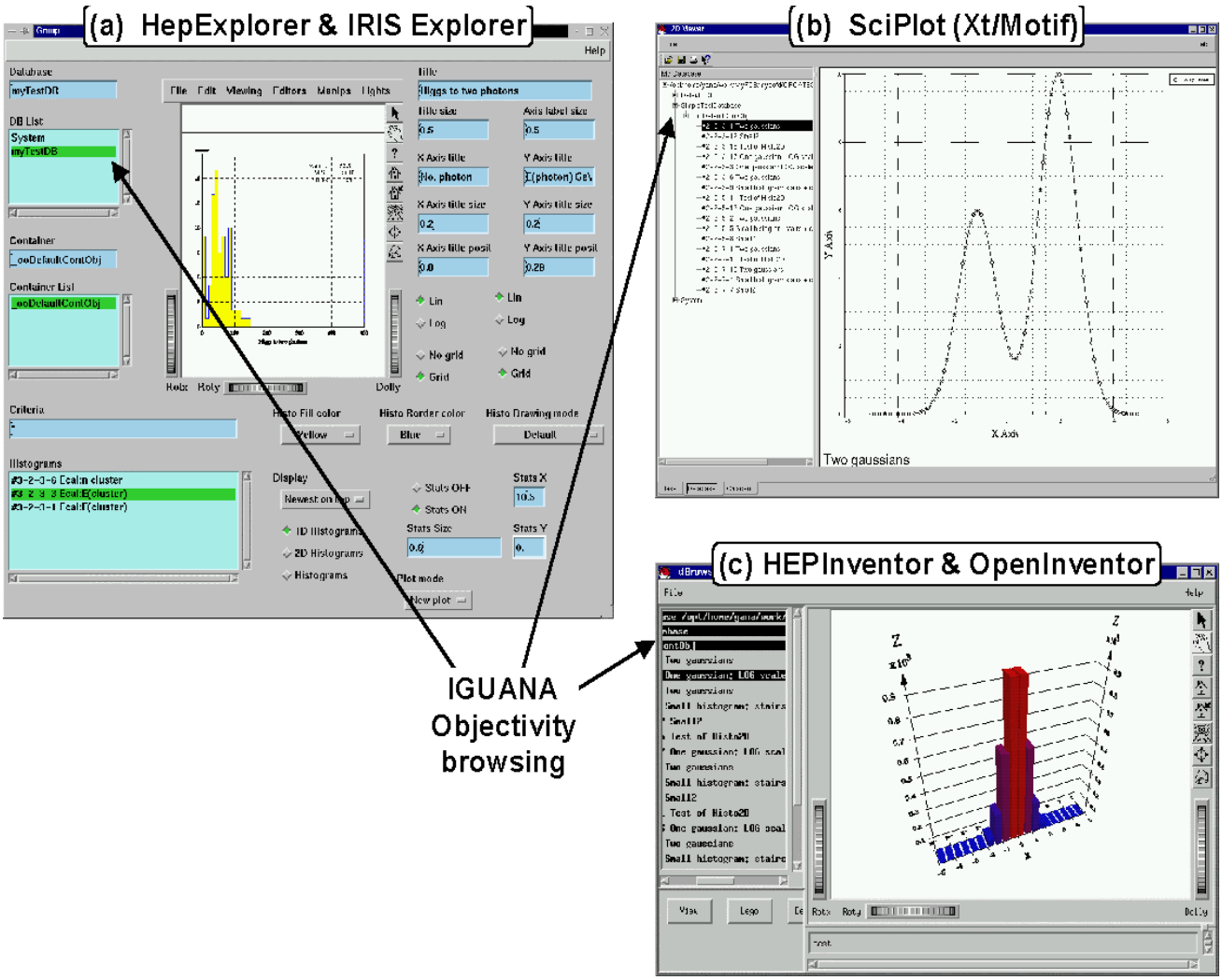

Figure 2.10: IGUANA Objectivity browsers prototypes to demonstrate various options for histogram plotters.

The above functionality has already been prototyped in IGUANA, with the exception of the dynamic loading of user code on demand, although the IGUANA approach has been predominantly GUI driven in contrast to the command line driven scripting environment of Lizard. The IGUANA experiences have been communicated with the Anaphe/Lizard developers through regular weekly meetings.

Since IGUANA may wish to exploit Lizard functionality from within other applications, an IGUANA prototype (IgPython) was developed in which a Python window was embedded within an MDI application. The prototype demonstrated the integration of a scripting language within a GUI application where all the functionality of Lizard is accessible from the command line of an "X-terminal" like IGUANA window.

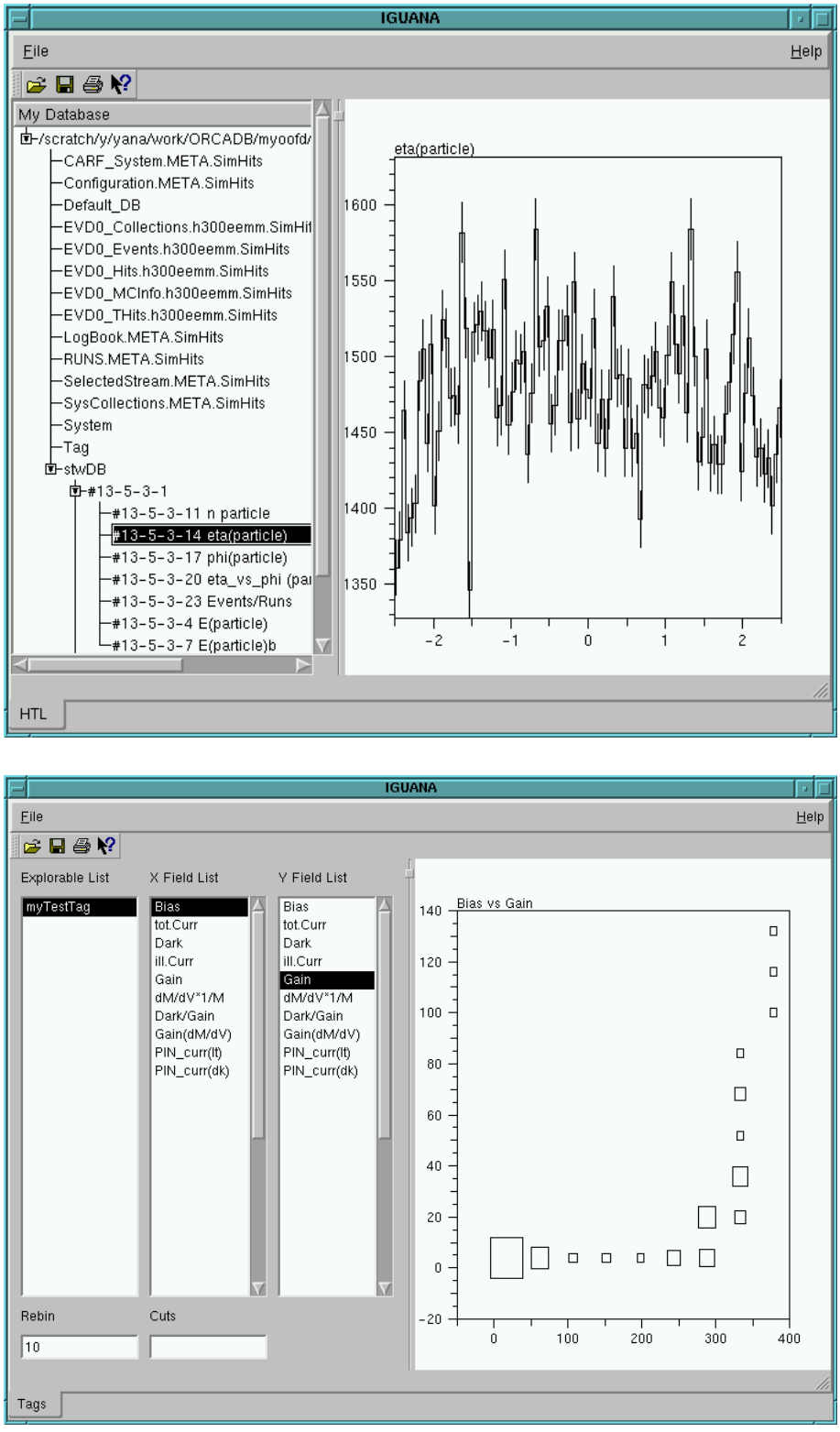

Figure 2.11: Prototype in which an IGUANA Objectivity browser is used to locate HTL histograms in a database federation and plot them using a plotter based on Qt and Qwt (top). The prototype of a tag browser with 2D histogram plotter (bottom).

Chapter 3

Software Infrastructure

This section describes the third deliverable for the IGUANA contribution to the "User Analysis Environment Functional Prototype" milestone of October 2000, which consists of a coherent software infrastructure including: a software repository; a multi-platform build, release, and distribution system; a comprehensive documentation system; and public versioned releases of both software and documentation.

Figure 3.1 summarises the tasks, packages, their origins, and the historical and proposed future usage of software associated to the "IGUANA Infrastructure" task. This task is further divided into sub-tasks, in particular "Code Management", "Code Quality and Testing", and "Documentation Systems" as described below.

IGUANA Infrastructure	*Package(s)*	*Origin*	*1999*				*2000*				*2001*				*2002*			
			Q1	*Q2*	*Q3*	*Q4*	*Q1*	*Q2*	*Q3*	*Q4*	*Q1*	*Q2*	*Q3*	*Q4*	*Q1*	*Q2*	*Q3*	*Q4*
					Functional Prototype Phase													
Code management																		
IGUANA repository	cvs	Public																
Build, release, distribution system	SCRAM	CMS																
Standalone remote distributor	OffsiteReleaser	IGUANA																
Code quality / checking																		
Dependency checking	IgNominy	IGUANA																
Documentation Systems																		
Reference documentation	doxygen	Public																
WWW pages builder	Ig_Documentation	IGUANA																
User guide builder	IgManuals	IGUANA																

Figure 3.1: Tasks, packages, their origins, and the historical and proposed future usage of software associated to the "IGUANA Infrastructure" task.

3.1 Code Management

Coherent code management is crucial to ensure that IGUANA software is usable by CMS collaborators on a variety of platforms both at CERN and in remote institutes. The IGUANA software organisation is consistent with the overall CMS software process [17], including the associated code management systems, version control mechanisms, and the code release, distribution, and build procedures.

3.1.1 Software Repository

The CVS system [18], which is prevalent in HEP and elsewhere, is used to manage the access, change, and version control of the IGUANA software repository. Within IGUANA the code is organised into sub-systems, as follows:

- *Cmscan* – generic interactive detector and event visualisation software;
- *Ig_Documentation* – documentation tools, templates, and high-level content;
- *Ig_Examples* – example programs;
- *Ig_Extensions* – extensions to non-IGUANA software packages;
- *Ig_Imports* – software which is imported to facilitate integration and distribution;
- *Ig_Utilities* – developer utilities such as the IgNominy dependency analyser.

Each of these sub-systems contains a number of smaller software "packages", each of which corresponds to a software library. The divisions between packages are chosen according to their expected functionality and also to minimise the number of external (non-IGUANA) libraries on which each one depends. This permits clients of IGUANA to deploy only those external libraries which are really required for a given application.

The `Ig_Imports` sub-system of IGUANA contains software which is imported to the repository and verified within the CMS environment by the IGUANA team. This strategy ensures mutual compatibility of the software versions required for the full IGUANA system and provides a convenient SCRAM-based mechanism by which external packages can be maintained and made available to the collaboration. The procedure of importing the software is automated wherever possible. For example, upgrades of the HEPVis library are performed by invoking a a script which exports the desired version from the Fermilab HEPVis repository and imports it into the IGUANA CVS repository.

3.1.2 Build, Release, and Distribution System

The CMS product SCRAM [7] is used to build and distribute the software. An important aspect of this task, which benefits greatly from the compatible ORCA activities, is the porting, technical verification, and support of CMS and external code on multiple systems. Additional makefiles have been added to the SCRAM configuration for the project which run the Qt Meta Object Compiler that is required to pre-process the header files of Qt-based packages. After a number of consolidating modifications in both SCRAM and IGUANA, a stable repository, release, and build procedure is now in operation.

Releases are made as required rather than with a fixed frequency. Each (immutable) release includes the source code, libraries, and binaries; the external configuration information and mechanisms for installing, and optionally re-building, the software at remote sites; results from the IgNominy dependency analysis (see section 3.2); and a fully-versioned set of documentation (see section 3.3). Releases are explicitly numbered x_y_z (rather than named *e.g.* new, dev, etc.), where: x is incremented for major releases, typically once or twice per year; y is incremented for minor releases, typically every few months, which may include package interface changes; z is incremented for bug-fix releases, typically every few weeks, which cannot include any interface changes.

3.2 Code Quality and Testing

CMS has discussed and adopted a comprehensive set of coding rules and guidelines. So far, however, there has not been a tool available which can automatically check all code for compliance to all rules, although this is seen as a priority item for the near future. Once such a tool is available, it will be deployed to improve the IGUANA code quality.

Already, each IGUANA package (library) has an associated test program. While this verifies that there are no major failures in in that package, it does not (necessarily) provide good coverage of all aspects of the code. This too is an area in which more systematic procedures will be required in the near future.

A key input for systematic testing is an understanding of the interdependencies of the various software components since these reflect the scope of the tests required. In order to analyse the dependencies of the IGUANA packages on each other and external software, the IGUANA team developed a utility known as IgNominy, as described below.

3.2.1 Software Dependencies and Metrics

The **IgNominy**[1] tool, developed within IGUANA, analyses package dependencies by examining a built source tree. It scans makefile dependency data for the actual header files which the compiler saw, source code, binaries and dynamic loader information for possible dependencies, resolves them and outputs a simple textual database. The information collected by IgNominy is then further processed by a number of presentation tools. The dependency diagrams are drawn using the external `Graphviz` package from AT&T.

IgNominy is used periodically during IGUANA development to ensure that the actual package dependencies are what they were designed to be. IgNominy may also be used to evaluate the modularity of external software in order to assess the ease with which it could be integrated with the rest of IGUANA. Recently, analyses of the Anaphe releases have provided invaluable feedback to the CERN/IT developers. It has proved invaluable in maintaining control over undesirable inter-package dependencies which tend to creep accidentally into developing code.

Results of running IgNominy are included in each IGUANA release in the `doc/deps` directory. The directory contains a subdirectory for each platform the release was built on, usually `linux` and `sunos`. The directory has an index page `index.html` pointing to the analysis results in a convenient way. This is hyper-linked from the IGUANA WWW pages for each release.

IgNominy also calculates some simple metrics following John Lakos: Large Scale C++ Software Design.[2] While IGUANA does not yet have a comprehensive testing framework, these metrics give a rough idea of how good the physical partitioning of a release really is. The metrics measure degree of *component dependency*, that is, the number of other (physical) components required to thoroughly and incrementally test a particular component. This also introduces the idea of *package levels*: packages on the lower levels must be tested before those on the higher levels (hence the incrementality).

The component dependency (which IgNominy calculates for packages, not individual classes) indicates how many other libraries must be linked into the test drivers of that package. The component dependency is *cumulative*: a package on level zero has no dependencies and hence cumulative component dependency (CCD) of 1 (it requires itself for testing) and the CCD of any other package is the sum of the CCD's of all the packages it uses. Essentially the CCD of a package tells how costly it is to test the package thoroughly. The CCD of the release is simply the sum of the CCD of all the packages in and used by the release. That is, the CCD IgNominy calculates includes the testing cost for all external software.

A couple of other numbers calculated by IgNominy are the average component dependency (ACD) and the normalised cumulative component dependency (NCCD). ACD is simply the release CCD divided by the number of packages involved, and indicates how many libraries are required on average in the tests. NCCD measures how the package dependencies compare to a balanced binary tree. NCCD less than one indicates that the package structure is more flat or horizontal than a balanced tree, and NCCD greater than one indicates that it is more vertical and/or tightly coupled. IgNominy reports the NCCD value with the minimum and maximum bounds: no interdependencies and total interconnectivity, respectively. A toolkit-like system such as IGUANA should have a NCCD less than 1.0, approximately 0.8 (assuming the package partitioning in reasonable). NCCD substantially greater than 1.0 would indicate that there may be significant cyclic physical coupling within the system. [3]

[1] **Ignominy**, *noun*: dishonour, disgrace, shame; infamy; the condition of being in disgrace,...

[2] `http://cseng.aw.com/bookpage.taf?ISBN=0-201-63362-0`

[3] Note however that Lakos has this to say about NCCD: NCCD is *not* a measure of the relative quality of the system. NCCD is simply a tool for characterising the degree of coupling within a subsystem. Increasing the number of components in a system could

Finally, IgNominy calculates another similar metric, the number of outgoing links for each package and the average over the system. Unlike ACD that measures all dependencies of a package, this number indicates the average number of *direct* dependencies.

3.3 Documentation Systems

Priority was given to the IGUANA documentation systems from the outset. All IGUANA documentation is stored in the same repository and versioned and released together with the code. Various types of documentation were recognised as being required, as described below.

- **A coherent set of WWW pages** (`http://iguana.cern.ch`)
  The IGUANA WWW pages are fully versioned and include all the documentation described below as well as various other items such as an IGUANA installation guide, a picture gallery, links to presentations, and mailing lists. All the WWW pages, apart from a single top-level page (see `http://iguana.cern.ch` and Figure 3.2) are generated for each release from files in the repository such that a complete and consistent set of appropriate pages is always available for each software release.
- **User guide / tutorial**
  A rather basic user guide / tutorial exists which describes laconically the way to get the IGUANA software installed and the example programs running. It is built automatically by combining a central LaTeX-based shell document with LaTeX files from various packages in the repository. The output is a combined document in PDF and PostScript formats. Figures in various formats (eps, epdf, png, etc.) are converted as required.
- **Reference documentation of the source code**
  This is built automatically from the source code using IGUANA tools which in turn rely on the public-domain doxygen product. This parses the source code and identifies embedded comments and their context (class, method, etc.). It then builds HTML documents which reflect the structure of the code and the content of the embedded comments.
- **Software release notes**
  Release notes, describing the main changes since the previous release, are available in HTML format for each new software release.
- **Architecture and design descriptions**
  There is currently no systematic procedure for documenting the architecture and design of IGUANA. This is acknowledged to be a weakness which should be rectified in the near future, in collaboration with the activities of the CAFE (CMS Architecture Forum and Evaluation) project.
- **IgNominy dependency analysis**
  The results of the IgNominy dependency analyses are generated for each release and made available on the corresponding WWW pages. The documentation includes an IgNominy overview, the dependency diagrams (gif and Postscript) for the complete system as well as individual sub-systems, the tabular summaries of incoming and outgoing dependencies, and the full log files.
- **Developers documentation**
  Developers documentation is available in HTML format to describe tasks such as, for example: making IGUANA releases; IGUANA setups on various platforms; building database federations for use with IGUANA; building the IGUANA software; creating all the various sorts of documentation including the fully-versioned set of WWW pages; and setting up new tools or versions of tools with SCRAM.

artificially reduce the NCCD. One way to do this is to eliminate completely valid reuse; this would likely not be an improvement. [...] **Minimising CCD for a given set of components is a design goal.** Reducing the CCD in a system of a given size is almost always desirable. Reducing the size (number of components) of a system is also desirable but not at the cost of introducing cyclic dependencies, inappropriately merging components, or creating unmanageably large translation units. [...] The precise numerical value of the CCD (or the NCCD) for a given system is not important. What is important is actively designing systems to keep the CCD for each subsystem from becoming larger than necessary.

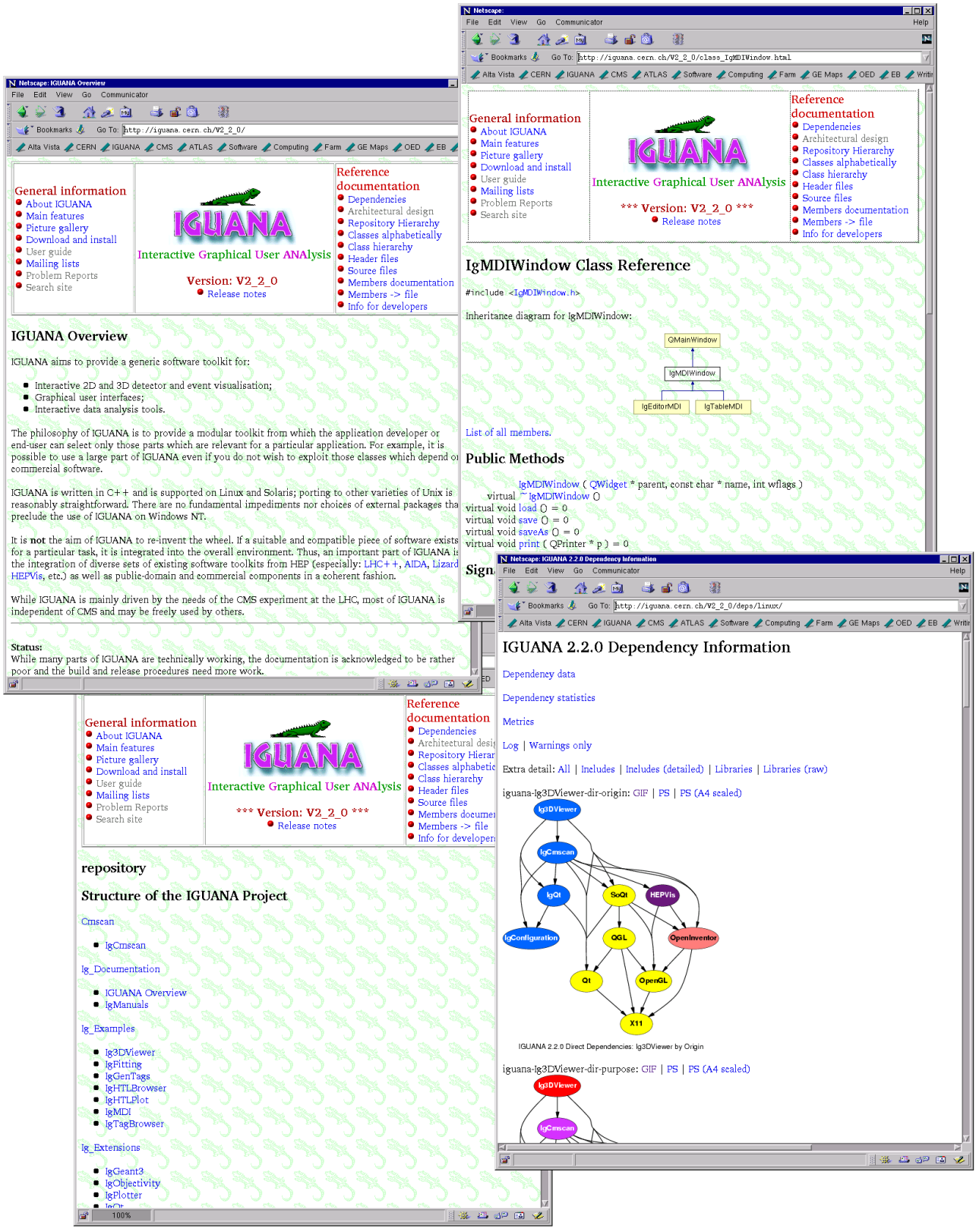

Figure 3.2: Sample pages from the IGUANA WWW documentation.

Chapter 4

Baseline Choice of Visualisation Technologies

This section describes the fourth deliverable for the IGUANA contribution to the "User Analysis Environment Functional Prototype" milestone of October 2000, which consists of a proposal for a baseline choice of visualisation technologies. These are *not* intended to remain static for the long term but are intended to constitute a well-defined and stable base from which to begin the next phase of software development.

The previous sections described the four main sub-tasks of the IGUANA project, together with a figure for each summarising the next level of sub-tasks, the associated packages, the period during which they have been used and the proposed future evolution, as follows:

- "Graphical User Interfaces" (Figure 2.2) with sub-tasks:
  - ➤ "GUI Toolkits"
  - ➤ "Widget Extension Libraries"
  - ➤ "Application Widgets"
- "Detector and Event Visualisation" (Figure 2.4) with sub-tasks:
  - ➤ "Underlying Graphics"
  - ➤ "Viewers"
  - ➤ "Detector Interfaces / Shapes"
- 'Interactive Analysis and Presentation" (Figure 2.7) with sub-tasks:
  - ➤ "Statistical and Numerical Analysis"
  - ➤ "Histograms and Tags"
  - ➤ "Plotting"
  - ➤ "Interactive Data Browsing and Analysis"
- "IGUANA Infrastructure" (Figure 3.1) with sub-tasks:
  - ➤ "Code Management"
  - ➤ "Code Quality and Testing"
  - ➤ "Documentation Systems"

Figure 4.1 summarises the packages associated to each of these tasks which are proposed for the baseline set with which to continue into the next "Fully Functional Software" development phase.

In line with the IGUANA philosophy, new packages will continue to be considered in future and added to the IGUANA software suite in future as appropriate. Similarly, packages in this baseline suite will be retired or replaced as and when deemed necessary.

Graphical User Interface	***Package(s)***	***Origin***
GUI Toolkits		
Low-level 2D graphics	X11	Vendor
GUI toolkit (+high-level 2D)	Qt	Public
Widget Extension Libraries		
IGUANA widgets: trees,tables,...	IgQt	IGUANA
Technical widgets	Qwt	Public
Flexible and editable table	Qt table module	Public
Application Widgets		
Sheet, document, window MDI	IgQtSheet...	IGUANA
Multi-document Interface (MDI)	Qt Workspace	Public
Detector&Event Visualisation	***Package(s)***	***Origin***
Underlying Graphics		
Low-level 3D graphics	OpenGL	Public/vendor
High-level 3D graphics	OpenInventor	Vendor / free
Qt / OpenGL mixing	QGL	Public
Qt / OpenInventor mixing	SoQt	Public
High-level 2D graphics	Qt Canvas	Public
Viewers		
3D Detector / Event display	IgCmscan	IGUANA
2D Detector / Event display	*to be named*	IGUANA
Detector Interfaces / Shapes		
GEANT3 interfaces	IgGEANT3	IGUANA
HEP OpenInventor extensions	HEPVis	HEP
GEANT4 interfaces	IgGEANT4	IGUANA
Interactive Analysis and presentation	***Package(s)***	***Origin***
Statistical/Numerical Analysis		
Mathematical library	NAG_C	Vendor
Minimiser (short/medium-term)	MINUIT	CERN/IT
Minimiser (medium/long-term)	NAG_C	Vendor
MINUIT / NAG_C wrapper	GEMINI	CERN/IT
C++ Fitting API to GEMINI	HepFitting	CERN/IT
Histograms / tags		
Histograms	HTL	CERN/IT
Tags / ntuple extension	HepODBMS	CERN/IT
Plotting		
1D and 2D plotting	Qplotter	CERN/IT
Surface function plotting	plot3d	Public
Interactive analysis		
"PAW replacement" application	Lizard	CERN/IT
Data browsers / utilities	*to be named*	IGUANA
IGUANA Infrastructure	***Package(s)***	***Origin***
Code management		
IGUANA repository	cvs	Public
Build, release, distribution system	SCRAM	CMS
Standalone remote distributor	OffsiteReleaser	IGUANA
Code quality / checking		
Dependency checking	IgNominy	IGUANA
Documentation Systems		
Reference documentation	doxygen	Public
WWW pages builder	Ig_Documentation	IGUANA
User guide builder	IgManuals	IGUANA

Figure 4.1: The tasks, packages, and their origins proposed for the baseline set of technologies to continue into the IGUANA "Fully Functional Software" phase.

Chapter 5

Proposed Project Evolution

This section describes the fifth deliverable for the IGUANA contribution to the "User Analysis Environment Functional Prototype" milestone of October 2000, which consists of a proposal describing how IGUANA should evolve into the "Fully Functional" software phase.

The three main IGUANA product categories, graphical user interfaces, detector and event visualisation, and interactive analysis and presentation, each require evolution. Similarly the IGUANA infrastructure needs attention. Our plans for these are described below. In addition to continuing work on these areas, we also identify two important new tasks. The first is to collect and analyse use cases. Secondly, having learnt much through the many prototypes we feel that the time is now right to feed this knowledge in addition to what we will learn from the use case analysis into a design of a coherent architecture that can integrate the whole of IGUANA together and with the rest of CMS software. The plans for these two new tasks are also described below.

In the larger context our plans follow the CMS software milestones as shown in table 5. The milestones reflect the iterative software development process consisting of four phases. The third phase beginning now involves the development of the prototypes into fully functional software, while the fourth is the preparation of the software for production and the exercising of the complete system in conjunction with the pre-production hardware systems. The relative priorities of all the following tasks will be adjusted to allow for CMS user and developer priorities and the availability of resources, particularly software engineers.

CORE SOFTWARE	1997	1998	1999	2000	2001	2002	2003	2004	2005
GEANT4 simulation of CMS (OSCAR)		❶		❷		❸		❹	
Reconstruction/analysis framework (CARF)		❶		❷		❸		❹	
Detector reconstruction (ORCA)			❶	❷			❸		❹
Physics object reconstruction			❶		❷		❸		❹
User analysis environment (IGUANA)			❶		❷		❸		❹

Table 5.1: Milestones for the CMS Computing and Software project (❶=proof of concept; ❷=functional prototype; ❸=fully functional software; ❹=production software).

5.1 Architecture Related Tasks

Having developed a large set of prototypes, many of which demonstrate how various existing technologies can be integrated together, we feel that this is right to begin developing a coherent architecture that can integrate all the tools in a modular fashion.

We expect this work also to be carried out in company with the efforts of the CMS CAFE project to define the CMS software architecture. CAFE will begin a high-priority task of collecting analysis use cases from the physics

community. These will need to be analysed and expanded to a sufficiently broad collection of key scenarios, which can in turn be used to evolve the requirements. This work will be done in close collaboration with IGUANA.

We propose to start this work in the following rather pragmatic fashion. Firstly, we will assess the prototypes we have and decide which parts of them may be appropriate for further development. Secondly, we will reconsider all the use cases to identify further requirements. Thirdly, we will develop a small kernel that can tie the existing functionality from the prototypes together. Fourthly, we will begin to develop a coherent data browsing application that integrates the various bits and pieces together. Fifthly, this application is extended in two directions: to give it ability to interface in a general way with CARF/ORCA and with generic analysis tools being developed by CERN/IT/API.

5.2 IGUANA Toolkit

Graphical User Interface

We expect that the graphical user interface components will require less of our attention during the next phase. We will continue to track Qt and related free widget components, and will migrate IGUANA components to new functionality as it becomes available and is required by us.

There are a number of potential GUI applications of IGUANA in the offline and test beam environments, from simple detector and event displays to monitoring and control interfaces and simple analysis systems. Some applications based on Qt have already been deployed successfully in the CMS H2B and X5B environments [19] and we plan to extend such activities.

Detector and Event Visualisation

CMSCAN—generic event visualisation
While basic 3D event visualisation has been completed and tested within ORCA, a number of enhancements are required. Many changes are non-essential but would improve usability, ranging from trivialities such allowing one to change colours to more complex ones such as supporting drag-and-drop. We also need to implement better feedback mechanisms between the graphics scene, the GUI, and the underlying data. Some of these will be covered by the architectural work described above. Dedicated 2D views need to be implemented, probably using Qt, as well as abstract or non-Cartesian views such as lego plots of energies as a function of pseudo-rapidity and azimuthal angle, fish-eye views, and so on.

Event visualisation with CARF and ORCA
The development and deployment of the CMSCAN event visualisation system for ORCA is ongoing. Following the imminent release of ORCA 4.4 we plan to work with ORCA developers to implement the visualisation of a complete set of reconstructed graphics objects; for some types this requires minor updates, and a handful still need to be implemented.

GEANT4 detector visualisation CMSCAN currently uses the GEANT3 geometry. We anticipate that the transition to GEANT4 geometries will be relatively straightforward due to the involvement of the IGUANA developers in the HEPVis project which has provided graphics software for GEANT4. The GEANT4 description of the CMS detector, when it is completed, will be visualised in both the OSCAR and the ORCA environments.

Interactive Analysis and Presentation

We propose to continue in this task with our current philosophy of a toolkit of loosely coupled software modules, from a wide variety of sources, with clearly defined interfaces and responsibilities.

We will continue to collaborate with other experiments and the Anaphe team. We expect that CERN/IT/API will further develop Qplotter such that it will cover, for example, most of our histogram rendering needs. We also expect them to concentrate more on many of the other non-graphics components such as histogramming (but

not display), object tags, statistical analysis, minimisation, scripting language(s), and a generic environment for analysis. We expect that we will continue to be able to use many of these as components in IGUANA, up to and including the generic analysis tools.

We propose that IGUANA continues the integration of these into the CMS environment and focuses in the beginning on providing the user community tools that they do *not* yet have. This includes in particular the following activities: much more convenient access to the databases and the data in them (especially the tags), facilitating and demystifing many of the daily database tasks, integrating the many presentation tools we already have into a more coherent environment, and interfacing the analysis tools to CARF, ORCA, OSCAR and other projects. We propose to begin this by evaluating and discussing different ways of browsing and interacting with data.

5.3 IGUANA Infrastructure

We consider increasing the quality and robustness of the software and the releases crucial. Nearly all the parts of the IGUANA infrastructure will need updates, from streamlining the release builds and access to more thorough (and partly automated) testing and more comprehensive manuals. Along with testing we will need several more example programs, in particular by making the examples more specific and well documented. Some of the architectural documentation, use cases and requirements will need to be migrated into a framework consistent with CAFE. Where appropriate these tools will be migrated to other CMS projects.

Bibliography

[1] L.Taylor. The CMS Software and Computing Project: Organisation, Tasks, Schedule, and Resources. CMS IN/1999-032.

[2] M. Schroder. CMS Detector Simulation Project OSCAR. CMS IN/1999-036.

[3] Z. Szkutnik. Minuit++ replacement for minuit and hep statistical tools; user requirements document. Minuit-URD-Draft, October 1997. (see also: `http://wwwinfo.cern.ch/ asd/lhc++/ requirements/ stable/URD/html/`).

[4] `http://wwwinfo.cern.ch/ asd/lhc++/ requirements/ special_functions.html`).

[5] V. Innocente. CMS Software Architecture: Software framework, services and persistency in high level trigger, reconstruction and analysis. CMS IN/1999-034.

[6] D. Stickland. CMS Reconstruction Software: The ORCA Project. CMS IN/1999-035.

[7] `http://cmsdoc.cern.ch/Releases/SCRAM/current/doc/html/`.

[8] `http://wwwinfo.cern.ch/asd/lhc++/`.

[9] M. Kalle Dalheimer. *Programming with Qt.* O'Reilly Verlag GmbH & Co. KG, Köln, 1999. See also: `http://www.troll.no/`.

[10] `http://www.accessone.com/~jql/perlqt.html`.

[11] `http://www.troll.no/qt/nsplugin.html`.

[12] `http://www.sgi.com/software/`. T. Davis, Jackie, Neider, M. Woo, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, Addison-Wesley, Massachusetts, (1993).

[13] J. D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, 1982.

[14] `http://www.sgi.com/Technology/ Inventor.html`.

[15] `http://www.troll.no/qt/opengl.html`.

[16] `http://www.sim.no/`.

[17] J-P. Wellisch. Status of Software Process Improvement in CMS. CMS IN/1999-033.

[18] See, for example: `http://www.sdsu.edu/doc/texi/cvs_toc.html`.

[19] L.Silvestris. Test-beam Software and Computing. CMS IN/1999-043.

[20] `http://www.troll.no/qt/xt.html`.

[21] `http://www.stack.nl/~dimitri/doxygen/index.html`.

Appendix A

IGUANA Software Prototypes

This Appendix contains brief descriptions of the main software components of IGUANA from the "Functional Prototype" phase, including for each one the purpose of the prototype, the functionality, comments, and finally the proposed action to take at the end of the functional prototype phase. Since it includes also discontinued software, it contain some useful lessons about unproductive lines of development as well as the final baseline choices.

IG-CO/A Xt and Motif Libraries

IG-CO/A-1:	**Ig_Extensions/IgXt**
Purpose:	Library of widgets based on Xt.
Functionality:	Various legacy widgets including the SciPlot widget for plotting 1D histograms, tree widgets, etc.
Comments:	Works but this essentially C code is inferior to more OO C++ widgets of Qt.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/A-2:	**Ig_Extensions/IgMotif**
Purpose:	Library of widgets based on Motif.
Functionality:	List tree widget can be used to graphically present the tree structures such as a database content or file system structure.
Comments:	Works but this essentially C code is inferior to more OO C++ widgets of Qt. Porting them from one system to another is a non trivial task. Not compatible with Windows. Use of Motif has been strongly discouraged mainly due to it's non-OO nature (Qt is far better). An additional disadvantage is the need to either purchase Motif or to try to use the free `Lesstif` version which currently has bugs.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/B Qt GUI Toolkit

IG-CO/B-1:	**Ig_Imports/qxt**
Purpose:	Support for the coherent use of legacy Xt/Motif widgets and Qt widgets in the same application.
Functionality:	Supports both Qt widgets used in an Xt/Motif (main-loop) application and vice-versa using the freely-available "qxt" extension from Troll Tech [20]. Various legacy Xt widgets were successfully wrapped including the mechanisms required for the consistent management of X events within either the Qt or the Xt/Motif context, as appropriate. For example, Figure 2.10 shows the Xt SciPlot widget for rendering histograms operating within a Qt histogram-browsing application.
Comments:	Although this works we do not recommend this approach to be used. Firstly, we consider Qt to be a far more appropriate base set of widgets which are functional, Object-Oriented, and available on all major platforms including MS Windows. Secondly, we are not aware of any significant amount of legacy Xt/Motif code.
Status/Plans:	**Completed. No future development nor support, except possibly bug-fix updates. Retain for use on "as is" basis. Defer support and distribution to another party if possible (within HEP or perhaps Troll-Tech).**

IG-CO/B-2:	**Ig_Extensions/IgQt**
Purpose:	Library of widgets based on Qt.
Functionality:	Miscellany of Qt extensions, including: multi-sheet MDI application classes; extended Qt ListView and ListViewItem, with multi-columns and check-boxes; tables, etc.
Comments:	Being truly OO Qt is easy to extend. Many items prototyped here subsequently seem to appear in the public domain (e.g. MDI, QTable, etc.) with a more polished implementation so this package will be dynamic and of ongoing utility.
Status/Plans:	**Continue development and support.**

IG-CO/B-3:	**Ig_Imports/qextmdi**
Purpose:	Imported package (qextmdi) which aims to provide a multi-document interface (MDI) based on Qt.
Functionality:	See also Ig_Examples/IgMDI.
Comments:	Will be obsoleted by new Qt (version $\geq$ 2.2) MDI implementation.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/B-4:	**Ig_Examples/IgMDI**
Purpose:	Evaluation of the imported package (qextmdi) which aims to provide a multi-document interface (MDI) based on Qt.
Functionality:	MDI modes are: mainframe with child views; all views are top level; tabbed pages (are not provided yet).
Comments:	Will be obsoleted by new Qt (version $\geq$ 2.2) MDI implementation.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/C Inter-operability of Qt, OpenGL, and OpenInventor

IG-CO/C-1:	**Ig_Imports/QGL**
Purpose:	Imported package (QGL) which aims to provide support for OpenGL rendering in a Qt environment.
Functionality:	The QGLWidget class is a widget for rendering OpenGL graphics. This class is a part of the Qt OpenGL Extension.
Comments:	There are several packages/libraries based on this extension such as QtAI: 2D, 3D graphics, etc. They have been evaluated.
Status/Plans:	**Ongoing use in CMS. Ongoing development and support external to CMS. Keep updated.**

IG-CO/C-2:	**Ig_Imports/SoQt**
Purpose:	Imported package (SoQt) which aims to provide support for OpenInventor in a Qt and QGL environment.
Functionality:	Supports mixing of OpenGL, OpenInventor, and Qt, in conjunction with QGL. Some fixes were required which have been fed back to the developers.
Comments:	Does not seem to be the mainstream activity of the developers therefore some exposure to a maintenance load should it become unsupported by them.
Status/Plans:	**Ongoing use in CMS. Ongoing development and support external to CMS. Keep updated.**

IG-CO/D Toolkits for 3D Interactive Visualisation

IG-CO/D-1:	**Ig_Imports/HEPVis**
Purpose:	Imported package (HEPVis) which aims to provide HEP specific extensions for OpenInventor.
Functionality:	Provides a number of Xt-based viewers and OpenInventor scene objects/kits such as detector elements and reconstructed event objects, which are based on the basic OpenInventor scene objects.
Comments:	Only a few scene objects are used. These are generally developed by the IGUANA team and then fed-back into HEPVis. The Xt/Motif-dependent components of HEPVis are excluded.
Status/Plans:	**Ongoing use in CMS. Ongoing development and support is external to CMS. Keep updated.**

IG-CO/D-2:	**Ig_Extensions/IgGeant3**
Purpose:	Design and implementation of a package which provides an interface between GEANT3 (Fortran) and the C/C++ API of OpenInventor for the visualisation of completely arbitrary detectors described by a GEANT3 geometry description.
Functionality:	Fortran routines are provided for walking down arbitrary GEANT3 geometry (GCVOLU common block) trees and requesting the creation of 3D OpenInventor geometry transformations and scene objects (shapes). Either full or partial traversal of the tree is provided through the GSATT options which are also used to control with arbitrary granularity the visibility of standard, divided, or repeated volumes. C++ code is provided to create OpenInventor shapes in World coordinates for the GEANT3 shapes, as follows: `hv_gbox.cc` (for: BOX, TRD1, TRD2, TRAP, PARA); `hv_gcyl.cc` (for: TUBE, CONE); `hv_pcon.cc` (for: TUBS, CONS, PCON); and `hv_pgon.cc` (for: PGON).
Comments:	Although this solution works, the interface with Fortran is (inevitably) not particularly clean nor is the GEANT3 code, some of which (`gdraw` and `gdraws`) were substantially cleaned and extended to create the `hv_gdraw` and `hv_gdraws` routines of this package.
Status/Plans:	**No new developments are foreseen for this code. Continued support to be provided as required. Allow for natural evolution towards obsolescence as GEANT4 applications gradually replace GEANT3-based applications.**

IG-CO/D-3:	**Ig_Examples/Ig3DViewer**
Purpose:	Design, implementation and evaluation of a generic Detector/Event display viewer component, based on SoQt, TGS Open Inventor and QGL, with full 3D rendering, interactive zooming, rotations, translations, and visibility control of the graphics tree.
Functionality:	Both the detector and the event are presented as a set of OpenInventor standard shapes. Selection from the list and the viewer with highlighting of the corresponding objects.
Comments:	Now consolidated into Cmscan/IgCmscan.
Status/Plans:	**Obsolete. No future development nor support. Remove from future releases.**

IG-CO/D-4:	**Cmscan/IgCmscan**
Purpose:	Design, implementation and evaluation of a generic Detector/Event display viewer, based on SoQt and QGL, with full 3D rendering, interactive zooming, rotations, translations, and visibility control of the graphics tree.
Functionality:	Basic viewer functionality works, as described in the main text.
Comments:	Would benefit from some re-examination of the design to allow extensions to other generic browsers such as 2D, text-based, or combinations of several browsers.
Status/Plans:	**Ongoing development and support. Consolidate with other examples.**

IG-CO/E Evaluation of HTL Histograms

IG-CO/E-1:	**Ig_Applications/IgHistos**
Purpose:	Design, implementation and evaluation of an interactive application which creates, fills, and plots of a set of *transient HTL* histograms (see also IgHTL).
Functionality:	Transient HTL histograms are created, added to a list of histograms. Selected histogram can be plotted.
Comments:	`tmake` might be useful for creating portable makefiles if there is a need to use IGUANA outside SCRAM. The current functionality is limited due to IgPlotter (see Ig_Extensions).
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/E-2:	**Ig_Extensions/IgHTL**
Purpose:	Design, implementation and evaluation of an interactive application which creates, fills, and plots of a set of *persistent HTL* histograms (see also IgHistos).
Functionality:	Database initialisation and transactions are hidden from a user, copy method provided to create a persistent HTL from a transient one.
Comments:	HepODBMS and CARF provide different way of session management. A User shouldn't be allowed to write anything in a System database. Naming is different. Invokes automatically the ooSession initialisation and sets the standard clustering hints.
Status/Plans:	**Completed. The session management has been migrated to the Utilities package in the ORCA repository as HTLHistogrammer. No future development nor support. Remove from future releases.**

IG-CO/F Evaluation of HepODBMS Tags

IG-CO/F-1:	**Ig_Extensions/IgObjectivity**
Purpose:	Convenience tasks associated to Objectivity.
Functionality:	Repackaging of legacy code from HEPExplorer (used with IRIS Explorer) which performed various tasks associated with Objectivity such as database initialisation, tags access, and tags manipulation. Retrieving generic tags attribute, cuts, etc.
Comments:	Equivalent and enhanced functionality to be provided by CERN/IT API group in future.
Status/Plans:	**Completed. No future development nor support. Remove from future releases once replacement from CERN/IT is completed.**

IG-CO/F-2:	**Ig_Examples/IgAPDTags**
Purpose:	Set of utility applications to read ASCII files of ECAL APD QA/QC data and create HepODBMS tags in Objectivity (see also IgAPD).
Functionality:	APD schema. Input: ASCII files.
Comments:	Starting from a simple data model it is possible to get to a very complicated one very quickly.
Status/Plans:	**Under development.**

IG-CO/F-3:	**Ig_Examples/IgAPD**
Purpose:	Set of utility applications to read ASCII files of ECAL APD QA/QC data and create *generic* HepODBMS tags in Objectivity (see also IgAPDTags).
Functionality:	Similar to HepODBMS example. Reads an APD-lab specific data file, parses it using ObjectSpace tokenizer and writes part of the information into an Objectivity database as generic HepODBMS tags.
Comments:	It's rather an HepODBMS example, then the IGUANA one. In other words, the example has been adapted for a specific case (APD lab file format).
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/F-4:	**Ig_Applications/IgLab**
Purpose:	Design, implementation and evaluation of a generic interactive application to analyse generic HepODBMS tags.
Functionality:	The tag database read is completely generic (e.g. that created by IgAPD). The application allows to browse the HepODBMS based generic tags, selected attributes can be plotted as 1D or as one vs another. Picking, zoom, and printing are provided by IgPlotter.
Comments:	HepODBMS based implementation allows relatively quick access to the tags, but not CARF compatible.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/G Prototype Histogram Plotters

IG-CO/G-1:	**Ig_Applications/IgCharts**
Purpose:	Evaluation of the imported widgets - QxPieWidget, QxColChart, QxLineChart, QxHistogram, which aim to provide various data plot presentation.
Functionality:	Several Qt widgets to provide different views of the ASCII data as pie-, line-, etc. charts. The data are read from the ASCII files in a simple format and presented with different widgets as pie-charts, line-charts, histograms, etc.
Comments:	Not a high priority.
Status/Plans:	**Completed. No future development nor support, except possibly bug-fix updates. Retain for use on "as is" basis.**

IG-CO/G-2:	**Ig_Imports/Qwt**
Purpose:	Evaluation of an imported package (Qwt) which aims to provide scientific graphics, notably 1D histogram plotting but also miscellaneous other widgets, for example scalars, dials, and thumb-wheels.
Functionality:	The Qwt library contains GUI components and utility classes which are primarily useful for programs with a technical background. Most of these widgets are used to control or display values, arrays, or ranges of type double.
Comments:	This Qt- and Qwt-based approach has been taken up by CERN/IT/API and will be further developed as the Qplotter package.
Status/Plans:	**Completed. No future development nor support, except possibly bug-fix updates. Retain for use on "as is" basis.**

IG-CO/G-3:	**Ig_Extensions/IgPlotter**
Purpose:	Design, implementation and evaluation of extensions to the Qwt library to provide addition *interactivity* not in the original package (see also IgQwt and IgTag3Plot).
Functionality:	Interactive plotter and legend: zoom, resize, picking.
Comments:	This Qt- and Qwt-based approach has been taken up by CERN/IT/API and will be further developed as the Qplotter package.
Status/Plans:	**Completed. No future development nor support. Remove from future releases once Qplotter has equivalent functionality.**

IG-CO/G-4:	**Ig_Examples/IgTag3Plot**
Purpose:	Design, implementation and evaluation of extensions to the Qwt library to provide addition *multi-zone support* not in the original package (see also IgQwt and IgPlotter).
Functionality:	Input: ASCII files. Select a zone (IgPlotter widget) to plot the input data.
Comments:	Dealing with a zone as a widget that can print itself would require to merge several postscript files in one page if a user would like to print a multi-zone page. This Qt- and Qwt-based approach has been taken up by CERN/IT/API and will be further developed as the Qplotter package.
Status/Plans:	**Completed. No future development nor support. Remove from future releases once Qplotter has equivalent functionality.**

IG-CO/G-5:	**Ig_Extensions/IgQwt**
Purpose:	Design, implementation and evaluation of extensions to the Qwt library to provide addition *rendering functionality* not in the original package (see also IgPlotter and IgTag3Plot).
Functionality:	Features added include the following: plotting a histogram as steps; brush and polygon filling; default background to white; axis with no offset; error bars drawing.
Comments:	This Qt- and Qwt-based approach has been taken up by CERN/IT/API and will be further developed as the Qplotter package.
Status/Plans:	**Completed. No future development nor support, except possibly bug-fix updates. Remove from future releases once Qplotter has equivalent functionality.**

IG-CO/G-6:	**Ig_Examples/IgTagPlot**
Purpose:	Evaluation of the use of the use of various IgQwt plot extensions (e.g. "x vs. y" line plot) using generic HepODBMS tags.
Functionality:	A list of generic HepODBMS tags, the tag attributes accessible for a selected tag.
Comments:	The tag analysis has been extracted and evolved in IgTagBrowser example. This Qt- and Qwt-based approach has been taken up by CERN/IT/API and will be further developed as the Qplotter package.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/G-7:	**Ig_Extensions/IgSciData**
Purpose:	Provision of a scientific 1D/2D vector class which includes both data values and also errors, for use when plotting.
Functionality:	Constructed from an HTL histogram.
Comments:	This (rather minor) package was always seen as a temporary solution required prior to the corresponding solution from CERN/IT API group. Now available as an abstract interface and concrete implementation from Anaphe.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/G-8:	**Ig_Applications/IgMap**
Purpose:	Evaluation of Qt's QPainter for simple 2D drawing for use in plots, event display, etc..
Functionality:	For this evaluation the data is simple 2D GPS positional data representing a GIS survey, read from an external ASCII file. Interactive plotter can parse the input data from moving a mouse in its window.
Comments:	Qt provides the rich text rendering and a generator of tags, but current implementation is limited to base any serious development on it. An external XML parser is needed if XML is to be used as an external data format. Really need a 2D equivalent of the OpenInventor scene graph.
Status/Plans:	**Completed. No future development nor support, except possibly bug-fixes. Retain for use on "as is" basis. Should evaluate the new QCanvas and re-examine the design of Qplotter.**

IG-CO/G-9:	**Ig_Imports/plot3d**
Purpose:	Evaluation of an imported package (plot3d) for plotting 3D functions (of 2 variables).
Functionality:	Plotting 3D surface of function $z = f(x, y)$. Support shaded and grid with hidden-line/surface removal modes, performant real-time surface rotation, printing resulting image, saving image to file. Parser features, for arguments x and y, include: Arithmetic operators: $+ - \times /$(); $r \equiv sqrt(x^2 + y^2)$; $a^b \equiv \|a\|^b$; $\|x\| \equiv abs(x)$; $[x] \equiv int(x)$; $\{x\} \equiv x - [x]$; functions: sin(x), cos(x), tan(x), exp(x), ln(x), lg(x), min($x1, x2, ...$), max($x1, x2, ...$), sqr(x), sqrt(x), asin(x), acos(x), atan(x), hsin(x), hcos(x), htan(x).
Comments:	Not a high priority. Suggested to CERN/IT/API to reuse it for Qplotter surfaces.
Status/Plans:	**Completed. No future development nor support, except possibly bug-fixes. Retain for use on "as is" basis.**

IG-CO/H Interactive Histogram and Tag Browsers

IG-CO/H-1:	**Ig_Examples/IgExDBrowser**
Purpose:	Design, implementation and evaluation of a Qt-based interactive application to scan collections of HTL histograms stored in Objectivity and plot them on-demand (based on Ig_Histos)
Functionality:	Does everything planned.
Comments:	One of the early prototypes that tested Qt compatibility with LHC++ (now Anaphe) components.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/H-2:	**Ig_Examples/IgHTLBrowser**
Purpose:	Design, implementation and evaluation of an interactive application (based on IgExDBrowser) to exclude write access to the database, and to extend the list tree view widget.
Functionality:	Improved functionality. No write access to Objectivity needed. List tree keeps type information about the objects.
Comments:	Watch out the HepODBMS implementation.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/H-3:	**Ig_Examples/IgHTLPlot**
Purpose:	Design, implementation and evaluation of an interactive application, based on IgHTLBrowser.
Functionality:	Database objects scanned and extended only when requested and added to the list tree. Selected HTL histograms can be plotted.
Comments:	Transactions should be hidden from a user. There should not be any built-in assumptions on where histograms can be stored, such as expecting them to be stored only in containers. Qt slots cannot be templated.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/H-4:	**Ig_Examples/IgFitting**
Purpose:	Design, implementation and evaluation of an interactive application which demonstrates the use of the HepFitting package for fitting an HTL histogram, with emphasis on the assessment of the ease-of-use of the user interface.
Functionality:	Setting the fit model, parameters, method, retrieving the result, contours, etc.
Comments:	During the exercise it became obvious that it is necessary to provide more OO interface for fitting packages. Current implementation of HepFitting needs to be revised. It's currently under development in CERN/IT/API (one full time FTE). The IFitting interface is being defined. Time scale: one month.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/H-5:	**Ig_Applications/IgAPDlab**
Purpose:	Design, implementation and evaluation of an interactive application to analyse data from the CMS ECAL APD QA/QC Laboratory. It aimed to be a test-bed for generic analysis components in an real environment albeit with a relatively simple data model.
Functionality:	The APD lab specific application browses persistent APD data objects (APD data schema, see above) in Objectivity. A mechanism to convert a selected object data to generic tags is provided. Plotting and printing of the data is possible as well as a table view (ASCII data). The application relies on a specific APD data schema. Though it can be easily separated (shared library loaded on request, etc.), for the time being it is a part of the application.
Comments:	Limited functionality of the IgPlotter puts restrictions on the application. Qt Architect was used to create the fit panel.
Status/Plans:	**Stable released version. More development needed to separate user-defined data from a generic analysis tool, to extend current functionality by switching to Qplotter, etc. Insufficient manpower to extend significantly or convert to a completely generic application. No future development nor support. Remove from future releases.**

IG-CO/H-6:	**Ig_Examples/IgTagBrowser**
Purpose:	Design, implementation and evaluation of an interactive application for browsing arbitrary HepODBMS tag collections, selecting variables to plot, applying simple cuts, creating and filling HTL histograms according to the selected variables and cuts, and plotting the resulting histogram on the screen.
Functionality:	1D histograms: plotted and printed; 2D histograms: printed only. The functionality is limited by a plotter.
Comments:	More sophisticated parser / scripting language clearly needed for cuts.
Status/Plans:	**Completed. No future development nor support. Remove from future releases.**

IG-CO/I Documentation Systems

IG-CO/I-1:	**Ig_Documentation/IgDoxygen**
Purpose:	Design and implementation of an automated system for building a fully versioned set of reference documentation of IGUANA from a variety of source code and other files.
Functionality:	Perl scripts analyse the various documentation files in the repository to create a complete version of all HTML pages which appear for a particular software release such that all documentation is, like the source code and libraries, fully versioned and self consistent. The reference documentation systems are based on moderately-formatted comments in the source code, in particular the header files describing the interfaces. The comments in the code are converted to fully hyper-linked documents using the `doxygen` [21] automatic documentation product. IGUANA has developed a Perl script, which can be invoked by SCRAM using the `scram b doc` command, to generate the full set of documentation. The script scans a given directory and creates html pages, based on the directory (or CVS repository) structure, the class inheritance structure, alphabetical ordering, etc. In addition appropriately cross-referenced LaTeX, Postscript, and PDF documents and Unix `man` pages are created.
Comments:	A concerted effort should be made throughout CMS to develop and deploy a single such system for all CMS software projects.
Status/Plans:	**Ongoing development and support. Develop and deploy for other CMS software projects as appropriate.**

IG-CO/I-2:	**Ig_Documentation/IgManuals**
Purpose:	Design and implementation of an automated system for building a fully versioned set of user documentation of IGUANA from a variety of files, producing a variety of output formats.
Functionality:	Makefiles build a document from each IgManuals subdirectory. The subdirectories contain the TeX files that make up the document as well as directions on how to collect TeX fragments from the packages' `doc` directories (for example, package-specific chapters of a user guide). The machinery produces PostScript (one-up and two-up double sided, plain and compressed), PDF (plain and compressed) and HTML versions of each document.
Comments:	Works as required.
Status/Plans:	**Ongoing development and support. Develop and deploy for other CMS software projects as appropriate.**

IG-CO/J Source Code Checking and Repository Maintenance

IG-CO/J-1:	**IgNominy**
Purpose:	Design and implementation of a tool to analyse our own and external packages for source and runtime dependencies.
Functionality:	For more details see the main text. IgNominy analyses a SCRAM-built tree and external libraries. Produces dependency data for sources (the header files the compiler actually included), binaries (shared libraries dynamic linker will load, libraries that will satisfy unresolved symbols in a component) as well as these both merged together. The information is presented in simple text output ordered by package, by domain in both forward and reverse directions. Integrated to SCRAM through the `scram b dependencies` command, and now part of the release procedure.
Comments:	Unwanted dependencies easily creep in as the packages evolve (hence this tool). The CMS tool DepUty could not be used for two reasons: it does not analyse external software nor does it handle the arbitrary repository hierarchies which are inevitably different than the CMS defaults for the imported packages. True dependency data is hard to gather from product descriptions. Useful to be run as a part of making a release, or whenever one wishes to ensure that no unwanted dependencies have crept in or remain after references to a package have been removed. Another tool to analyse unnecessary `include` directives would be useful.
Status/Plans:	**Retain and support.**

IG-CO/J-2:	**checkBuildfiles.sh**
Purpose:	A tool to check for the existence of hardwired INCLUDE paths, library paths, or AFS dependencies in the SCRAM BuildFile's of IGUANA.
Functionality:	A simple shell script works as required.
Comments:	Crucial prior to release since it catches possible violations of the distributed (non-CERN environment) software release due to forgotten de-bugging or prototyping hacks.
Status/Plans:	**Retain and support. Make the execution of this task an automatic part of the release procedure.**

IG-CO/J-3:	**deleteCVSDirectory.sh**
Purpose:	A tool to recursively delete locally and within CVS a directory and all its contents.
Functionality:	A simple shell script works as required.
Comments:	In general, CVS is poor at managing directories making cleaning a repository (e.g. "moving" files) painful. We could benefit additional tools on top of CVS or possibly a replacement (perhaps `subversions`?).
Status/Plans:	**Retain and support.**

IG-CO/J-4:	**OffsiteReleaser**
Purpose:	Light-weight limited functionality distribution and build for non-CMS off-site institutes for quick evaluation purposes.
Functionality:	Creates compressed tar files (one per platform) of source, pre-built libraries, and a trivial makefile to build executables.
Comments:	
Status/Plans:	**Completed. Support.**

IG-CO/J-5:	**IgConfiguration**
Purpose:	Provision of a C++ header file containing switches and fixes required for platform independence.
Functionality:	A file `Architecture.h`, based substantially on the corresponding ORCA file, provides the required configurability.
Comments:	Should be included everywhere in source files (not in header files). This should really be a common solution with ORCA and other CMS projects once the cross-project configuration management supports this in an effective manner. This is a potential problem for imported third-party packages.
Status/Plans:	**Retain and support. Ensure that IGUANA adopts, and helps support, a CMS-wide mechanism once it is in place.**